ATARI® *Portfolio*™

Technical Reference Guide

First Edition
September 1989

CONFIDENTIAL

The information in this guide is proprietary and confidential. This guide and the information herein may be used only in accordance with the written agreement under which it was supplied.

Copyright © 1989 and 1990, Distributed Information Processing (DIP), Ltd.

All rights reserved. Copyright in this Portfolio Technical Reference Guide, the software contained in the Portfolio, and all technical drawings of and relating to the hardware design ("the Copyright Material") is vested in Distributed Information Processing Limited ("DIP"). Accordingly no part of the Copyright Material may be reproduced, stored in a retrieval system or transmitted in any form or by any means without the prior written permission of DIP.

Atari, the Atari logo and Portfolio™ are trademarks or registered trademarks of Atari Corporation. IBM PC is a trademark of International Business Machines Corporation. Lotus 1-2-3 is a trademark of Lotus Development Corporation. MS-DOS is a trademark of Microsoft Corporation. Turbo C, Sidekick are trademarks of Borland Corporation. XTALK is a trademark of Microstuf Corporation.

Atari Corporation
1196 Borregas Avenue
Sunnyvale, CA 94089-1302

CONTENTS

1 Technical Overview of Portfolio 1-1

2 Hardware

2.1 System Description 2-1
2.2 System Memory 2-4
2.3 Memory Cards 2-6
2.4 Custom ASIC Chip 2-8
2.5 Power Supply Unit 2-9
2.6 Portfolio Expansion Port 2-11
2.7 Peripheral Design Issues 2-17
2.8 LCD Display 2-21

3 Software

3.1 General Description 3-1
3.2 Differences Between Portfolio BIOS and IBM PC BIOS 3-4
3.3 System Specific BIOS 3-6
3.4 Differences Between MS-DOS and Portfolio DOS 3-23
3.5 Device Drivers and Peripheral Software 3-24
3.6 Memory Cards 3-28
3.7 Screen Handling 3-30
3.8 Power Management 3-31
3.9 Special File Formats 3-33
3.10 IBM PC Development System 3-37

4 Peripherals

4.1 Portfolio Serial Communications 4-1
4.2 Smart Parallel Interface File Transfer Protocol Description 4-3
4.3 IBM PC Card Drive 4-4
4.4 EPROM Writer Adaptor Boards 4-5

Appendixes

A Example .RUN Program.....A-1
B Diagram of Portfolio Character Set.....B-1
C Example Peripheral Design.....C-1
D Memory size and assignment.....D-1

1 TECHNICAL OVERVIEW OF PORTFOLIO

The Atari Portfolio is the first product that provides the functionality of a standard desktop PC in a package which can fit into a pocket.

The main requirements for the Portfolio technology are that the product should be pocketable, compatible, have low power consumption and above all be inexpensive to purchase.

The Portfolio is the size of a video cassette (VHS) and weighs less than one pound (450 grams).

The Portfolio provides a high degree of software compatibility with the industry standard desktop microcomputer, the IBM PC. This is achieved by supplying a PC-compatible BIOS, MS-DOS-compatible operating system as well as Lotus 1-2-3 file-compatible spreadsheet. The Portfolio also provides an expansion bus connector which allows peripherals to be connected to the product.

The Portfolio is inexpensive to manufacture as the software is provided on and runs from ROM and with the large scale integration of system logic, using an ASIC, the overall component cost and size are reduced.

The Portfolio uses credit card-sized memory cards instead of magnetic disks, and a LCD display. These components have a low power consumption and hence the product can use the consumer standard AA batteries and achieve a long battery life.

This Technical Reference Guide describes the Portfolio technology in detail and provides the necessary information for a third party to develop hardware and software applications for the Portfolio. This document only provides information specific to the Portfolio technology. If you want information on the standard IBM PC hardware, BIOS, or MS-DOS then the following publications should provide the required information:

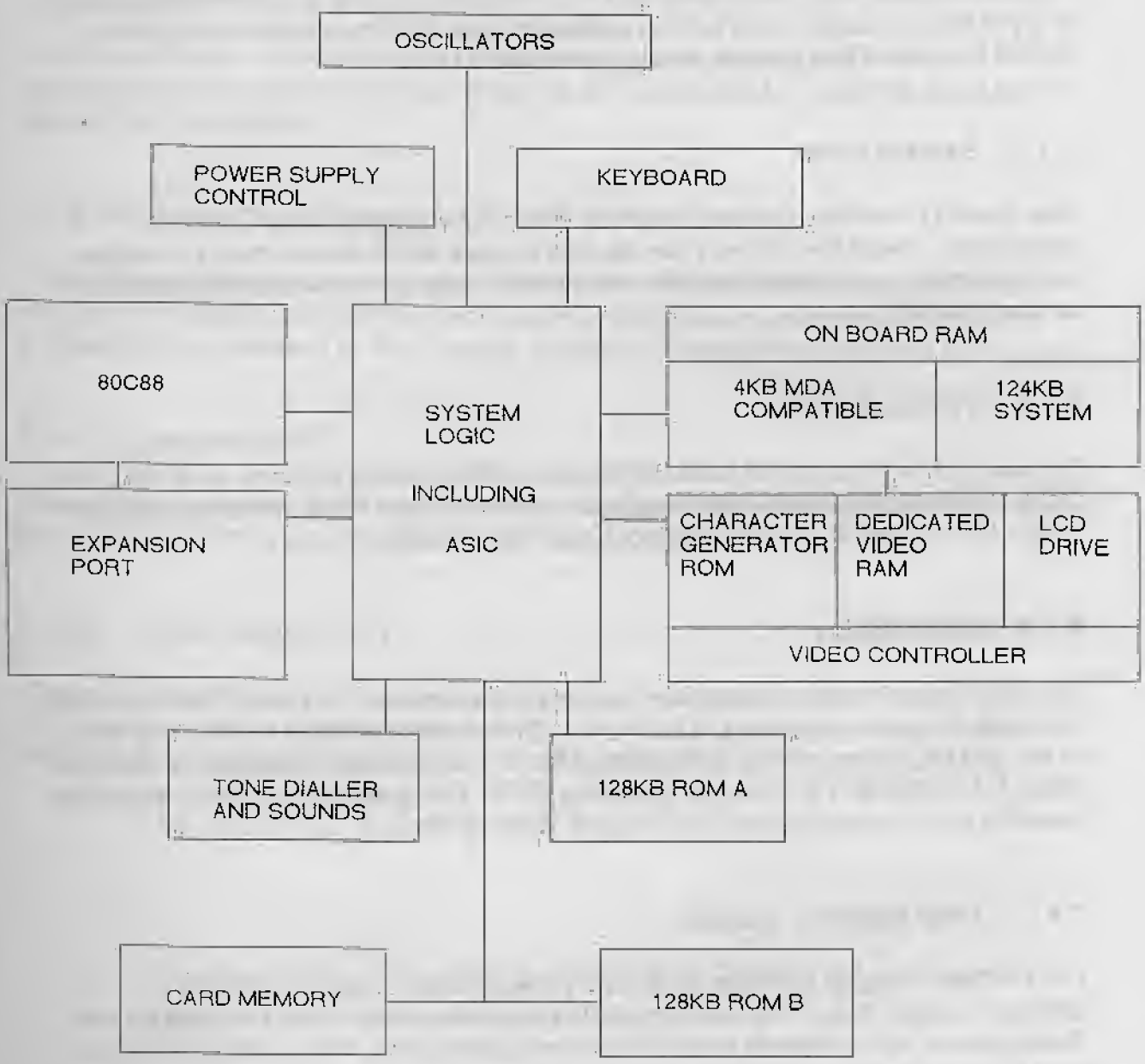
IBM Personal Computer Publications:

Technical Reference (BIOS and Hardware)
Disk Operating System (PC-DOS)

Microsoft Press Publications	ISBN Reference
Peter Norton Programmer's Guide	0-914845-46-2
The MS-DOS Encyclopedia	1-55615-049-0
IBM ROM BIOS (Quick Reference Series)	1-55615-135-7
MS-DOS Functions (Quick Reference)	1-55615-128-4

2. HARDWARE

2.1. System Description



BLOCK DIAGRAM OF PORTFOLIO SYSTEM

2.1.1 Microprocessor Unit

The Portfolio uses an 80C88 MPU, the same processor as the original IBM PC. It is the CMOS static version which allows the MPU clock to be halted when no processing is taking place and hence power consumption is kept to a minimum. The Portfolio is faster than the original IBM PC, the clock running at 4.9152MHz instead of 4.77MHz. However, the Portfolio processor uses minimum mode, so bus lock cannot be used. (See section 2.6 for more details.)

2.1.2 System RAM

The current Portfolio uses four 32 KByte Static RAM chips, giving a total of 128KBytes. These have a very low standby current which allows them to maintain their contents for extended periods with minimal drain on batteries.(See section 2.2 for more details.)

2.1.3 System ROM

The current Portfolio has in total 256 KBytes of ROM which contains all of the BIOS, DOS, command processor and application software. This ROM currently comprises of two 128 KByte chips. (See section 2.2 for more details.)

2.1.4 LCD Display

The LCD is a 240 x 64 pixel display. It is driven by a set of LCD screen drivers which are controlled by a graphics LCD controller. The screen behaves in the same way as an IBM PC monochrome text screen (MDA). The controller also uses a dedicated Video RAM chip and a character generator ROM. For graphics it is pixel compatible provided the PC-compatible BIOS is used. (See sections 2.2.2, 2.8 and 3.7.)

2.1.5 Tone Dialler + Sound

The Portfolio speaker is driven by a Dual Tone Multiple Frequency (DTMF) telephone dialler chip. This produces all the necessary dual tones required for tone dialling plus a set of melody tones for musical applications. The keyboard click also uses this circuit. (See section 3.3.1.)

2.1.6 ASIC

This circuit contains most of the system logic. (See section 2.4 for more details.)

2.1.7 Keyboard

The Portfolio uses a 63-key QWERTY 'switch-matrix' keyboard. The ASIC generates a set of physical scan-codes which are translated by the BIOS to IBM PC-compatible scan-codes.

2.1.8 Memory Card Connector

The Portfolio contains a memory card connector on the side of the product. Credit card-sized memory cards can be inserted into this connector, allowing for data and programs to be accessed by the Portfolio software. (See section 2.2.4, 3.6.)

2.1.9 Expansion Port

On the right-hand side of the product there is a 60-pin connector which provides the necessary signals for various peripherals. (See section 2.6, 2.7 and 3.5.)

2.1.10 Power Supply Unit

This supplies all the power required in the system. It produces various supply lines. The circuit includes a switching regulator that steps up the voltage from 3 AA cells to 5V. The regulator may be switched off. (See section 2.5 and 3.8 for more details.)

2.2 SYSTEM MEMORY

2.2.1 Memory Map

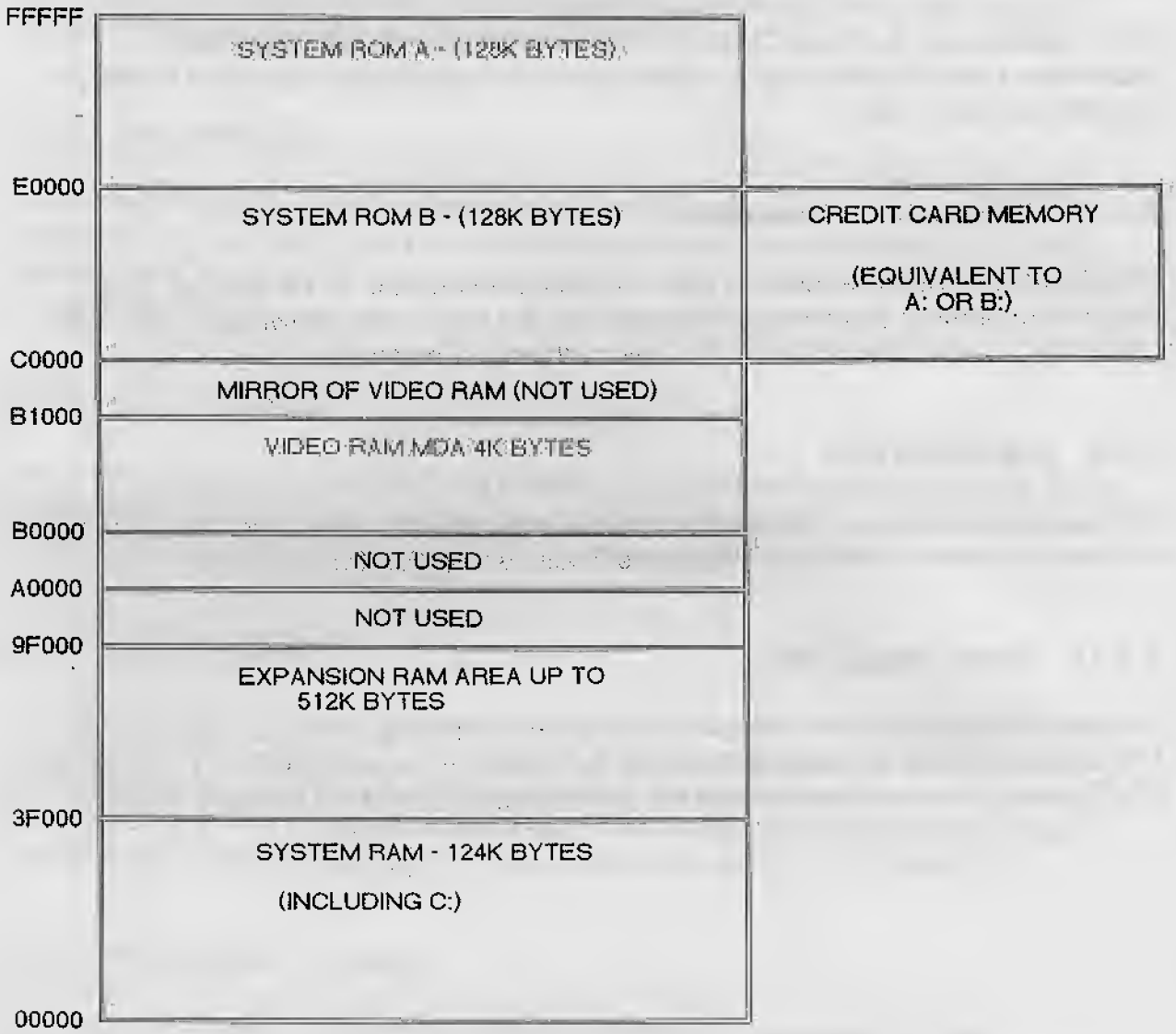


Diagram of Portfolio Memory Map

(all addresses in hex)

2.2.2 RAM

There is a total of 128 KBytes of on-board RAM provided with the Portfolio.

4 KBytes of this RAM are mapped to B0000h to make up the video RAM which is compatible with the IBM PC MDA screen. This gives a total system RAM of 124 KBytes.

The system RAM can be expanded up to 636 KBytes by use of memory expansion peripheral(s).

The Portfolio allows the user to have an internal RAM disk (known as C:) which can be user configured. This RAM disk uses the top of the system RAM.

2.2.3 System ROM A

This contains the BIOS, operating system and some of the application software. The reset vector sits at FFFF0h. This ROM cannot be mapped out of the memory map.

2.2.4 System ROM B

This contains the rest of the application software. This ROM may be switched out of the memory map and replaced by either the internal memory card or an external memory card on a peripheral. The BIOS disk services would normally perform this switching function. (See section 3.6.)

2.3 Memory Cards

The Portfolio uses credit card-sized memory cards which are specially designed for the Portfolio. There are similar memory cards available from other vendors. DO NOT use these cards with the Portfolio as they may harm the card and the Portfolio.

These come in three main types: RAM, OTPROM and Mask ROM. (See below for explanation.)

The cards are formatted to look like MS-DOS disks. It is possible to run a program directly from a card and hence reduce the amount of system RAM required. (See section 3.6.3 for more details.)

2.3.2 RAM cards

The RAM cards are currently available in three main sizes: 32, 64 and 128 KBytes.

The cards are made up of Static RAM and each card contains a lithium back-up cell. This cell will maintain the data on a card when it is not in a Portfolio for a year or more.

2.3.3 OTPROM cards

The One Time Programmable ROMs cards that are currently available are 64 and 128 KBytes. They are read-only cards and would typically be used for holding fixed data or software. They can be programmed in a standard EPROM programmer like a normal PROM (see section 4.4).

2.3.4 Mask ROM

These cards are "factory programmed" and have a low unit cost. This makes them suitable for issuing mass production software. Currently available only as 128 KByte option.

2.3.5 Future Card Sizes

The Portfolio BIOS contains support for ROM and RAM cards of greater than 128 KBytes. If these become available, they will be made up of 128 KByte pages with a page register at offset 000Ah. It is imperative that NO application software uses this memory card location, no matter what the card capacity.

2.3.6 Memory Card Pin-out

Below is a pin out of typical memory cards. Differences between the various card types are highlighted. (Pin 1 is on the right with the connections up and pointing to you.)

Pin	COMMON	RAM	OTPROM			Mask ROM	
			32k	64k	128k		
1	A16						
2	A15						
3	-	VBB	VPP	NC	VPP	NC	
4	A12						
5	A7						
6	A6						
7	A5						
8	A4						
9	A3						
10	A2						
11	A1						
12	A0						
13	D0						
14	D1						
15	D2						
16	GND						
17	D3						
18	D4						
19	D5						
20	D6						
21	D7						
22	CE						
23	A10						
24		OE	OE	OE/VPP		OE	OE
25	A11						
26	A9						
27	A8						
28	A13						
29	A14						
30		WE	NC	NC		PGM	NC
31	VCC						
32	CDET						

Notes:

NC	No internal connection.
VCC	Operating supply: 5 Volts.
GND	Signal ground
CDET	This is the small pin (internally connected to GND), used to detect presence of card.
Ax	Card address line x.
Dx	Card data line x.
VBB	Card battery voltage.
PGM	OTPROM program line. 0V in program mode
VPP	OTPROM program voltage. 12.5V Program mode, 5V normally
OE	Low to indicate a read cycle.
WE	Low to indicate a write cycle.

2.4 Custom ASIC Chip

The Portfolio custom ASIC chip provides most of the necessary system logic. It is a gate array implemented using silicon gate CMOS technology, which allows for very low power and high speed operation.

This Application Specific Integrated Circuit (ASIC) is used to generate all the select lines for the memory, memory cards and other system blocks. It also contains several system control functions. These functions are controlled using a set of registers which control the various parts of the system such as memory chip size.

2.4.1 System Clock

The clock is 4.9152MHz, with a 50% duty cycle produced by a crystal oscillator. The clock can go in to a stop mode. A custom chip interrupt will cause the clock to restart.

2.4.2 Timer

The system timer tick count is generated from a 32768Hz crystal oscillator which will generate an interrupt every 1 second or every 128 seconds.

2.4.3 Keyboard Controller

The keyboard controller will scan an eight by eight push to make key-switch matrix. A pressed or released key will cause an interrupt. The processor will obtain the scan code from a control register.

2.4.4 Interrupt Handler

This controls the critical error for the memory cards, keyboard and tick count interrupt. This is extended outside the ASIC to allow for external peripheral interrupts.

2.4.5 Soft Contrast for LCD

A control register holds the contrast value for the LCD display.

2.5 Power Supply Unit

The Portfolio has several power supply lines and control lines. These are used for various purposes and have different power characteristics as explained below. They are all available on the expansion bus.

When batteries and a power supply are connected to the Portfolio simultaneously, the Portfolio will be supplied by the higher voltage ("initial source").

(See section 3.8 for software issues.)

2.5.1 Power Modes

i) NO POWER MODE
This is the state when no initial source is connected to the Portfolio (e.g. changing batteries). If an initial source has been supplied and then removed, the system RAM will be backed up by an internal capacitor.

ii) OFF MODE
This is the state the Portfolio goes into when the 'OFF' command is used. The custom chip and RAM are powered directly from the initial source.

iii) STANDBY MODE
This is the state that the Portfolio will be in while waiting for a key press. The whole system is powered from the output of the internal five volt regulator. However, the system clock CCLK is halted in order to stop the processor and save power.

iv) RUN MODE
This the state in which the Portfolio is actually processing. The whole system is powered from the output of the five volt regulator and the system clock CCLK is running, thus causing maximum power usage.

2.5.3 VCC (Memory Card supply voltage)

This line follows 5VS. It is designed to be used by an external memory card so that plugging in and pulling out a card will not cause spikes on 5VS. This line should not be used for any other purpose.

2.5.2 5VS - Five Volt switched supply line

This is the output of the five volt regulator. During STANDBY and RUN modes, this line will supply five volts. At any other time this line will float low. Peripherals may be designed that use this supply.

5VS is capable of supplying up to 40 mA at 5V +/- 5% to a peripheral. This assumes that the main unit is taking maximum power. The Portfolio will run correctly outside the 5% supply tolerance; however, this is not recommended.

Use of 5VS by a peripheral will decrease the unit's battery life. Also, since alkaline batteries develop a voltage drop (due to internal resistance) the low battery warning will occur when the batteries are less depleted than if the peripheral was not plugged in.

2.5.4 VRAM (Memory Power Supply)

This is the supply for the system RAM:

STANDBY and RUN 4.5V
OFF initial source voltage - 0.5V
NO POWER current voltage across a capacitor.

During NO POWER Mode the voltage will decay, so care should be taken that no current is taken from this line or the system RAM could be corrupted. When the cold reset switch is pressed this line is pulled to GND through a small resistor.

2.5.5 VEXT (External voltage)

This is the external power supply voltage connected directly to the external jack socket. This enables peripherals with their own power source to make use of the Portfolio external power supply. It is possible to supply the Portfolio via this connection. However, care should be taken to avoid external power supply conflicts.

2.5.6 BATD (Battery detection signal)

This control signal is used to isolate system RAM from the rest of the circuit when the batteries are removed. It would normally carry the initial source but when the initial source is removed, it is pulled to GND. This line could be used by a peripheral to access the initial source.

2.6 Portfolio Expansion Port

The Portfolio uses a 60-pin expansion connector which can take custom designed peripherals. (See section 4 for more details on current range of peripherals.)

2.6.1 Expansion Port Connector Pin-out

Location of pin 1 - If you are looking into the Portfolio expansion port then the top pin on the right is 1 and the bottom right is 2.

ABUF	. 1	2.	5VS
REDY	. 3	4.	VCC
BCOM	. 5	6.	NCC1
NMD1	. 7	8.	WAKE
DTR	. 9	10.	DEN
PDET	.11	12.	IINT
CCLK	.13	14.	MRST
HLDA	.15	16.	HLDO
IACK	.17	18.	CDET
IOM	.19	20.	A19
A18	.21	22.	A17
A16	.23	24.	A15
A14	.25	26.	A13
A12	.27	28.	A11
A10	.29	30.	A9
A8	.31	32.	VRAM
HLDI	.33	34.	ALE
GND	.35	36.	NMIO
OA7	.37	38.	OA6
OA5	.39	40.	OA4
OA3	.41	42.	OA2
OA1	.43	44.	OA0
AD0	.45	46.	AD1
AD2	.47	48.	AD3
AD4	.49	50.	AD5
AD6	.51	52.	AD7
EINT	.53	54.	NRDI
VEXT	.55	56.	EACK
BATD	.57	58.	NWRI
5VS	.59	60.	BBUF

2.6.2 Explanation of expansion pin names

This section explains the functions of the expansion port. It assumes a knowledge of 80C88 minimum mode. Detailed Timing for relevant signals can be found in a microprocessor data sheet, ideally OKI MSM80C88ARS-2.

REDY output
This line indicates to the CPU that the custom chip is ready. This line is active high.

VCC output
This is the Credit Card power supply.

BCOM output
This is the communications select line, used for peripheral implementations. It is low if I/O locations 807X are being accessed. This signal is active within 100nS of I/O address being valid (see section 2.7).

NCC1 output
This is the external credit card chip select line. It is low if the external credit card is selected. See BCOM for timing.

NMD1 input
This is the external credit card detect line. It goes low to indicate that a card is plugged in.

DTR input/output
This is the 80C88 data direction signal. During CPU HOLD this line may be driven.

DEN input/output
This is the 80C88 data enable signal. Low indicates a data cycle. During CPU HOLD this line may be driven.

PDET input
This is the peripheral detection line. It should be tied high on a terminating peripheral that has a PID. (See section 2.7 for more details.)

IINT output
This is the internal interrupt request line to the CPU (INTR). It goes high to indicate an interrupt request.

IACK input/output
This is the 80C88 interrupt acknowledge line (INTA). It goes low to request an interrupt vector after an IINT. During CPU HOLD it may be driven by external hardware.

EINT input
This is the external interrupt request line. It may be driven high by external hardware on a terminating peripheral to request an interrupt. This interrupt line has lower priority than the on board interrupts. This signal is level triggered.

EACK output
This is the external interrupt acknowledge line from the Portfolio. It goes low to request an interrupt vector after an EINT. It follows \INTA on the processor, but is delayed by up to 40nS.

CCLK output
This is the main processor clock (4.9152MHz, 50% duty cycle). Since the clock pauses when no processing is taking place, dynamic logic should not use this line. It may be used for synchronising peripheral logic. During halt mode this line is high. This signal is only available to terminating peripherals.

MRST output
This indicates system reset. MRST will normally be high, except when a terminating peripheral is installed. The terminating peripheral will experi ence a short reset when inserted. If a terminating peripheral is installed then MRST goes high to indicate system reset. MRST will remain high at any time the reset key is pressed. It will also go high when the main computer system powers up. Under these conditions MRST will remain high for over 300mS.

HLDI input
This is the hold request line and will drive HOLD on the 88C88. It may be driven high by external hardware to requisition the system bus.

HLDO output
This is the 80C88 hold request line (HOLD). HLDI should be used to request a HOLD.

HLDA output
This is the 80C88 hold acknowledge line (HLDA). It goes high to indicate that the bus is now free. This state will be called CPU HOLD.

WAKE input
This line is used by a peripheral to wake up the main computer when it is powered down. This line is set low to request wake up. Wake up can be confirmed by waiting for a falling edge on MRST. It will take 300-400mS for wake-up to be confirmed. When wake up is confirmed, the wake input should be released.

CDET input
This signal is tied low to indicate to the main computer that an external credit card drive is present.

IOM input/output
This is the 80C88 memory access select line. If high then a I/O cycle is taking place, if low then a memory cycle. During CPU hold this line may be driven.

VRAM output
This is the RAM power supply. It will backup RAM when the batteries are removed, therefore any current taken from this line should be µAs.

5VS output
This is the switched 5V output. There are two 5VS lines.

GND output
Signal ground line.

VEXT output
This is the external power supply line.

ALE input/output
This is the address latch signal from the CPU. It latches the address bus on its falling edge. During CPU HOLD this line may be driven.

A8-A19 input/output
These are the upper part of the address bus from the CPU. During CPU HOLD these lines may be driven.

AD0-AD7 input/output
These are the multiplexed address/data bus from the CPU. During CPU HOLD these lines may be driven.

OA0-OA7 output
These are the lower latched address lines.

NRDI input/output
This is the 80C88 \RD signal. It goes low to indicate a CPU read cycle. During CPU HOLD this line may be driven.

NWRI input/output
This is the 80C88 \WR signal. It goes low to indicate a CPU write cycle. During CPU HOLD this line may be driven.

BATD output
This is the detect line for the batteries. It goes low if the batteries are removed without a power supply being present. This can be used to prevent accidental corruption of RAM.

ABUF/BBUF input
These are insertion detection pins. A terminating peripheral should have these lines connected to the adjacent 5VS line. (See section 2.7 on peripheral design.)

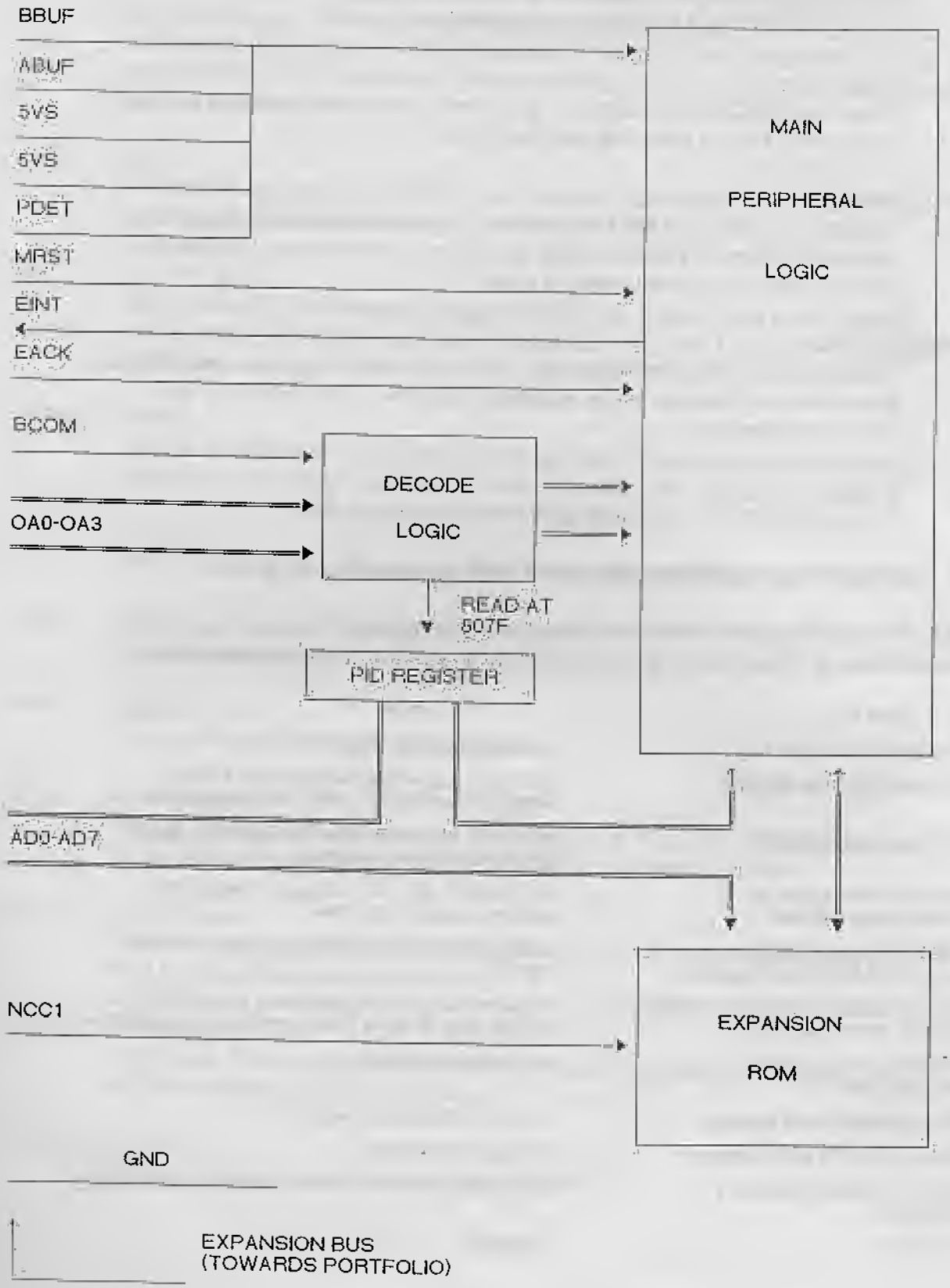
NMIO output
This is the 80C88 non-maskable interrupt request line.

2.6.3 Comparison between IBM and Portfolio expansion bus

The IBM PC and Portfolio expansion buses are analogous; however, the implementation of these buses are very different. See the comparison below:

	IBM PC	Portfolio
a)	I/O is partially decoded.	I/O MUST be fully decoded.
b)	A0-A19 are latched address	OA0-OA7 are latched address A8-A19 are address lines DIRECT from the processor.
c)	D0-D7 are buffered data	AD0-AD7 are multiplexed address/data lines DIRECT from the processor.
d)	IRQ2-IRQ7 are inputs to the interrupt controller.	EINT/EACK allow connection of peripheral with an interrupt controller.
e)	IOR/IOW/MEMR/MEMW are MAX mode bus control signals	NRDI/NWRI/IOM are MIN mode bus control signals.
f)	DRQ1-DRQ3/DACK0-DACK3/AEN/TC are DMA control signals	No analogous signals, however, enough control signals exist to allow DMA control on a peripheral.
g)	I/O CH RDY inserts wait states for slow I/O.	No analogous signal.
h)	ALE is address latch enable.	ALE is address latch enable
i)	OSC is 14.31818 MHz Clock.	No equivalent signal.
j)	CLK is 4.77 MHz, 33% duty cycle clock	CCLK is 4.9152 MHz, 50% duty clock which halts.
k)	IO CH CK	No signal.

TYPICAL TERMINATING PERIPHERAL



2.7 PERIPHERAL DESIGN ISSUES

There are two types of peripheral that can be connected to the Portfolio. These peripherals either continue the system bus ("Through Peripheral") or not ("Terminating Peripheral"). Different considerations are required for designing these types of peripherals. Appendix C illustrates an example peripheral design. (See section 3.5 for software issues.)

2.7.1 Terminating Peripherals (see diagram)

A peripheral of this type signals its presence to the Portfolio by having PDET tied high. If PDET is high then the Portfolio expects to see a Peripheral Identifier(PID). This is read from I/O location 807Fh. The returned number must be in the range 40h-7Fh. Please note that identifiers under 40h and over 7Fh are reserved for use by DIP and must NOT be used by non-DIP applications.

A read at I/O location 807Fh should NOT be used by any peripheral in ANY way other than as stated above.

A terminating peripheral can have an expansion ROM which contains support software for the peripheral. This eliminates need for software to be supplied separately. The chip select for this ROM should be connected to NCC1. At various times during a boot sequence, the ROM will be checked for an identifier. If this is present then the software contained on the ROM will be executed. (See section 3.5 for more details.)

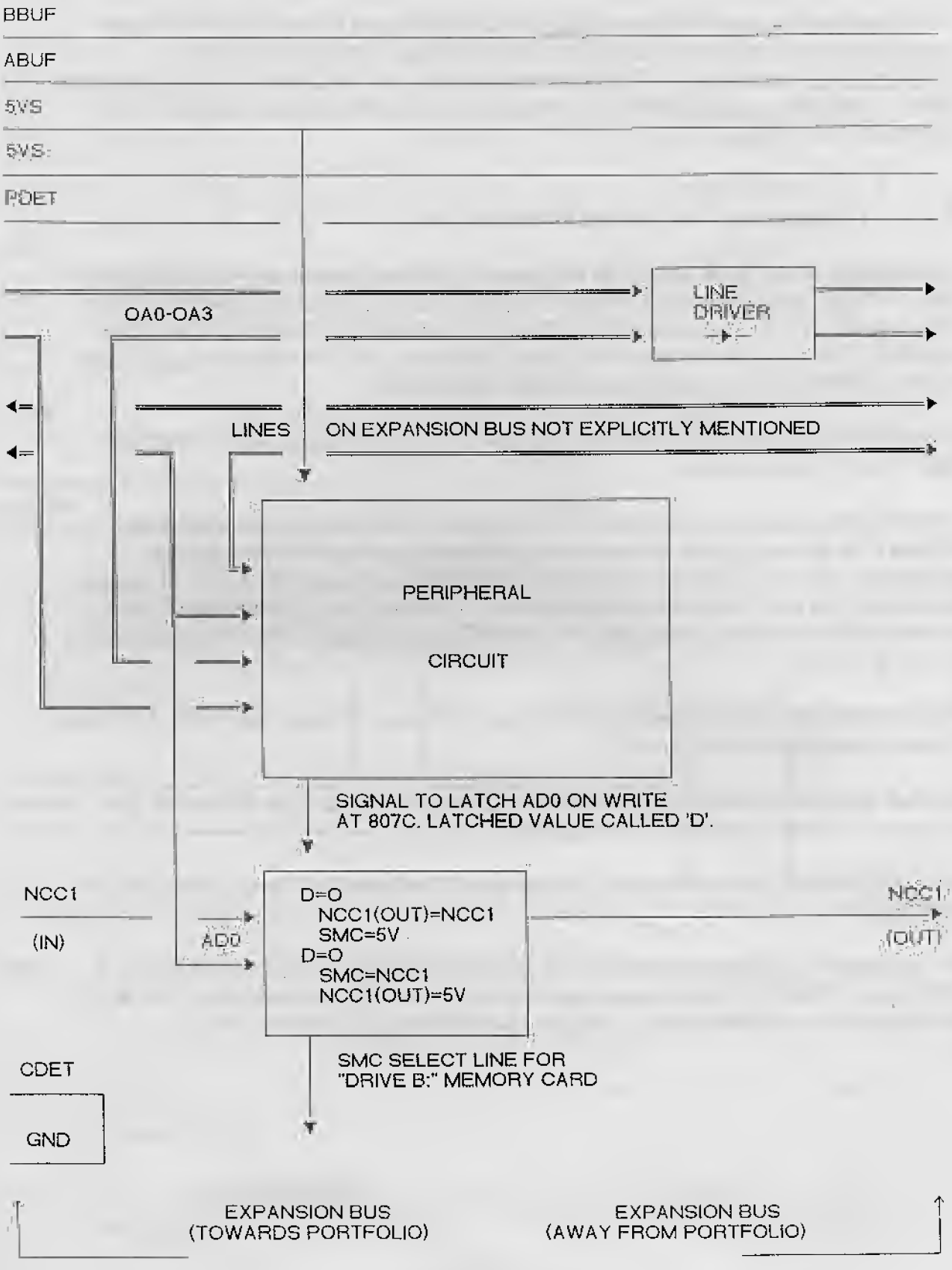
The Portfolio has only limited expansion bus buffering. To make best use of this the following rules should be obeyed.

i) ABUF and BBUF should be tied to the adjacent 5VS lines. This will cause the processor to hold while a connector is half in.

ii) Only OA0-OA3 should be used. BCOM should be used for higher addressing on peripherals.

iii) Peripherals using the external interrupt facility should be reset by MRST into a state where interrupts are disabled until the vectors are set up correctly. This is to prevent spurious interrupts occurring before the interrupt vector is set up.

TYPICAL THROUGH PERIPHERAL



2.7.2 Through peripherals (see diagram)

On these peripherals the system bus is continued so that further peripherals can be connected to the system. For example, a memory expansion unit would be this type of peripheral. In order that terminating peripherals will operate correctly the following recommendations should be taken into account when designing "through" peripherals.

i) ABUF and BBUF should not be connected to 5VS, but brought straight through the peripheral.

ii) If OA0-OA3 are used on the peripheral, they should be buffered before the "through" connector.

iii) PDET should not be connected to 5VS, but brought straight through the peripheral.

iv) The I/O locations 8070-807F should not be used so as to provide compatibility with terminating peripherals using these locations (such as DIP serial and parallel peripherals). 807Ch can be used as stated in vi) below.

v) "Through" peripherals risk crashing the system bus as virtually no buffering exists. It is therefore recommended that these peripherals are only inserted or removed from the Portfolio when powered down.

vi) To ensure that ROM extensions on terminating peripherals function correctly, through peripherals which contain a memory card interface must supply logic that follows the following rules:

* A write of zero to I/O 807Ch will cause NCC1 to be directed to the through expansion port.

* A write of one to I/O 807Ch will cause NCC1 to be directed to the peripheral memory card interface.

2.7.3 Allocation of Peripheral ID (PID) bytes

The PIDs have currently been allocated as follows:

PID	PERIPHERAL
00h	Communication Card
01h	Serial Port
02h	Parallel Port
03h	Printer Peripheral
04h	Modem
05-3Fh	Reserved
40-7Fh	User Peripherals
80h	File-Transfer Interface
81-FFh	Reserved

For custom user peripherals a specific PID can be allocated by contacting the Atari Portfolio Product Manager in writing, describing the use of the peripheral.

2.8 LCD Display

The Portfolio uses a 240x64 pixel LCD display which uses the "Super-twist" technology. This corresponds to 8 lines of 40 characters text display.

The circuit includes a graphics LCD screen controller with dedicated screen RAM chip and character set ROM, used in such a way as to be compatible as possible with an IBM Monochrome Display Adapter (MDA). (See section 3.7 for more details.)

The LCD circuit has the following characteristics:

* Full IBM PC Extended character set (see Appendix B)

* Virtual 80x25 MDA screen page with various screen modes

* PC-BIOS compatible pixel Set/Reset for graphics

* Each character is implemented as an array of 6x8 pixels

* Software controlled contrast

* Block or underline cursor

Note: Screen text attributes and various cursor modes are not supported by the Portfolio.

3 SOFTWARE

3.1 General Description

3.1.1 Overview

The Atari Portfolio software is contained on ROM and predominantly executes from ROM, and hence minimizes the use of RAM. This software provides as much PC compatibility as possible given the hardware constraints. (See sections 3.2 and 3.4 for BIOS and DOS comparisons.)

This software also includes some more advanced features which enable the Portfolio to be used more effectively in a portable environment than a standard PC. Most of these software features are accessed using Interrupt 61H, the Atari Portfolio specific functions. (See section 3.3.1.)

To aid development of application software for the Atari Portfolio which require the use of these specific functions there is a TSR (Terminate and Stay Resident) Emulator program for the IBM PC. This program emulates most of the functions. (See section 3.10 for more information.)

3.1.2 Portfolio Programming

The Portfolio obeys IBM's own programming guidelines for PC compatibility, however these are a lot more flexible than the industry-standard definition of a 'clone' PC.

Most 'well-behaved' PC programs run with no problem on the Portfolio, provided that they do not go below the BIOS to directly use the hardware. The main development issues are the screen size and memory capacity. Below are the various points to take into consideration when developing a program for the Portfolio.

SCREEN - (See also section 3.7.)

The Portfolio has a 40 column by 8 line text display which uses video RAM at the same address as the PC Monochrome Display Adaptor (MDA) and uses the same character set. However the Portfolio LCD controller does not support text attributes such as bold, underline and reverse or the various cursor sizes. If you want to use the Portfolio graphics facility then use the standard BIOS pixel read and write interrupt.

MEMORY - (See also section 2.2.)

The Portfolio has an internal memory disk C: which can be configured in 8KB intervals, minimum 8KB. This leaves a maximum of 116 KByte usable RAM of which 10 KBytes are used by the operating system and BIOS. Therefore it is recommended that programs should not use more than 100KBytes of system RAM. If you want to use the built-in 'pop-up' applications with the external program then allow for some free RAM (minimum of 17 KBytes).

MEMORY CARDS - (See also section 3.6.)

These memory cards appear to a DOS program like a standard floppy disk. The Portfolio has DOS resident all of the time and therefore does not need to boot from a disk. If you want to automatically boot into a program then you can put AUTOEXEC.BAT on a memory card, overriding C:\AUTOEXEC.

RS232/SERIAL - (See also section 4.1.)

The only compatible method for accessing the serial port is through the BIOS. However most off-the-shelf serial programs go directly to the hardware.

KEYBOARD- (See also section 3.2.1.)

The Atari Portfolio supplies full IBM PC scan-code compatibility provided access is through the BIOS. In other words it is possible to generate every keypress or combination that a standard PC can generate (SHIFT, CTRL, ALT, NUM PAD). It is also possible to generate other non-PC key combinations necessary for functions such as contrast and switching off.

POWER - (See also section 3.8.)

For power conservation, it is recommended that programs are designed which do not poll the keyboard continuously.

ADVANCED

There are also more advanced features which enable custom programs for the Portfolio to perform more sophisticated tasks, such as running programs directly from the memory cards (section 3.6), peripherals with built-in software on ROM (section 3.5), language information and access to the built-in tone dialler.

3.1.3 Troubleshooting

Running well-behaved standard off-the-shelf PC Programs:

* Make sure that the DISPLAY SETUP (see user manual) is set to Static PC for External programs.

* If the program writes directly to Video RAM then ensure that DISPLAY REFRESH is set to KEYBOARD or FAST TIMED, whichever is more appropriate.

* Endeavor to allocate enough system RAM.

Although many popular programs are 'well-behaved' there are also many programs which directly address the hardware. This can cause a problem on the Portfolio as the IO addresses are different. The most common of these incompatibilies occur with the keyboard and hardware interrupts. The Portfolio does not have a Programmable Interrupt Controller (PIC) or a dedicated keyboard controller, therefore some programs which access these such as Basic and XTALK will not function correctly. The Portfolio also uses a different Timer Tick than a PC which affects some 'dirty' programs such as Sidekick. Another hardware area that differs on the Portfolio is the use of the speaker, which should be accessed using the BIOS.

3.2 Differences Between Portfolio BIOS and IBM PC BIOS

For the purposes of this document, Portfolio BIOS is defined as the program which communicates between the DOS and the hardware. (See recommended books in section 1 for more information on the standard PC BIOS.)

There are a few differences between Portfolio BIOS and the standard IBM PC BIOS. These are generally in areas where the hardware differs to such an extent that complete compatibility is unobtainable. For example, in the Video Services (Int 10H) the Portfolio only has two screen modes; 80 by 25 Text and 240 by 64 Graphics.

3.2.1 Interrupt differences

The following list highlights the main differences between the DIP BIOS and IBM PC BIOS:

Int 09H * Keyboard
The Portfolio keyboard is not at the same IO address as a standard IBM PC, therefore any program which requires the keyboard to be at port 60H will not work correctly.

Int 10H Video Services
Service 00H, Mode 07 to 0AH are supported, but only in Text or Graphics mode. Service 01H, Cursor size is set to either block or u/line. (See section 3.7.)

Int 13H Disk
The Portfolio has modified Memory Card/Disk services 0 to 05H and 83H. (See section 3.3.2 for more details.)

Int 15H Extended
No Extended services are available.

Int 16H Keyboard
Only service 0, 1, 2, 4 are supported.

Int 18H BASIC
Not supported.

Int 1AH Clock
Only services 0-07H supported.

Int 1CH Timer tick
Invoked less frequently than IBM PC (see section 3.3.1).

3.2.2 Portfolio BOOT procedure

On a COLD boot (batteries removed, COLD Reset switch pressed, then batteries replaced), the BIOS executes a limited Power On Self Test (POST) to verify system integrity. This will destroy data in system memory (both programs in the Transient Program Area and those on internal drive C:). The Portfolio system then performs BIOS and DOS initialization before jumping to the COMMAND processor. This will always reset the machine unless there is a hardware fault.

On a Hardware WARM boot (WARM Reset switch pressed or batteries replaced without pressing the COLD Restart switch), the Portfolio performs BIOS and DOS initialization before jumping to the COMMAND processor.

On a Software WARM boot (Ctrl-Alt-Del on keyboard), the sequence of operations is similar to those for a Hardware warm boot. The difference between the two is that a Hardware warm boot also resets the ASIC and Processor which may be necessary if the interrupts have been disabled because the keyboard will not recognize user key presses.

3.3 System Specific BIOS

3.3.1 Int 61H - DIP extended BIOS services

Function	Description
0H	Service Initialization
7H	Format Credit Card Memory (CCM)
8H	Get size of Internal disk
9H	Format Internal disk
BH	Determine if CCM present
DH	Get Screen size
EH	Get/Set Screen mode
FH	Get/Set Cursor mode
10H	Get/Set virtual screen position
11H	Move virtual screen position
12H	Screen refresh
15H	Sound generation
16H	Melody tone
17H	Dial number
18H	Mute states
19H	Get Serial port parameters
1AH	Get Peripheral ID byte
1BH	Set Peripheral ID byte
1CH	Preset Peripheral IO data
1EH	Get/Set Clock tick speed
1FH	Get-key/Tick Screen refresh
20H	Disable revectoring of Int 9H
24H	Get/Set ROM space state
26H	Get/Set Power State
28H	Get/Set Language
2CH	Get BIOS version number
2DH	Turn system off
2EH	Enable/Disable status line
30H	File transfer via smart cable

Note: There are other reserved Int 61H services which are used internally by the Operating system. It is not recommended that these services are invoked by applications software, as they may be modified or deleted in future versions of the software.

Fn 00H Service Initialization 3.10

Parameters:
AH 00H
Returns:
None

Note: This service should be called once only as part of its initialization by any application program that intends to use any Int 61H function calls.

Fn 07H Format Credit Card Memory 2.3, 3.6, 3.32

Parameters:
AH 07H
AL Drive number (0 or 1)
Returns:
CF Set if error during format
AH Error code (See INT 13H)

Note: Drive number 0 selects drive A:, and drive number 1 selects drive B:. This service should not be used to format the internal disk (drive number 2).

Fn 08H Get size of Internal disk 3.3.2

Parameters:
AH 08H
Returns:
AX Segment Address of disk
BX Size of disk in Kbytes

Fn 09H Format Internal disk 3.3.2

Parameters:
AH 09H
BX Size of disk in Kbytes
Returns:
If CF=1
BX Maximum size possible (K)

Note: The system is rebooted if successful. All files on drive C: will be lost

Fn 0BH Determine if CCM present and valid 2.3, 3.3.2, 3.6

Parameters:

AH	0BH
AL	Drive number (0 or 1)

Returns:

CF=0	Card present and correct
If CF=1	
AH	Error code (See Int 13H)

Note: This can be used to determine if a valid CCM is in the specified drive. Drive number 0 selects drive A:, and drive number 1 selects physical drive B:.

Fn 0DH Get screen size 2.8, 3.7, 2.1.4

Parameters:

AH	0DH

Returns:

AX	Physical screen size
DX	Logical screen size

Note:

AH/DH	Row number
AL/DL	Column number

Fn 0EH Get/Set screen mode 2.1.4, 2.8, 3.7

Parameters:

AH	0EH
AL=0	Get mode
AL=1	Set mode
DL	New mode

Returns:

If AL=0	
DL	Mode
If AL=1	
DL	Old mode

Note: The mode is changed by setting one of the following mode bits in DL:

Clear bits (00H)80 by 25 mode
bit 0 (01H) 40 by 8 mode
bit 1 (02H) Tracked mode
bit 7 (80H) Graphics

These bits are mutually exclusive. When changing to 40 by 8 mode, if the cursor position or virtual screen origin is off the screen, then the virtual screen origin will be set to (0.0), the Screen cleared and cursor homed.

Fn 0FH Get/Set Cursor mode 2.1.4, 2.8, 3.7

Parameters:

AH	0FH
AL=0	Get mode
AL=1	Set mode
BL	New Cursor mode
AL=2	Force mode

Returns: If AL= 0

BL	Cursor mode

If AL> 0

BL	Old Cursor mode

Note: Cursor mode is as follows:

0	Cursor off
1	Underline
2	Block

Force mode automatically sets the BIOS cursor size to reflect the Keyboard Numlock state.

Fn 10H Get/Set virtual screen position 2.1.4, 2.8, 3.7

Parameters:

AH	10H	
AL	0	Get position
AL	1	Set position
If AL=1		
DH	Row number	
DL	Column number	

Returns:

If AL=0		
DH	Row number	
DL	Column number	

Note: The virtual screen position is the top left origin of the 40 by 8 window on the logical screen.

Fn 11H Move virtual screen position 2.1.4, 3.7

Parameters:

AH	11H	
AL	Number of lines to move cursor	
DL	Direction to move cursor	
	1	Up
	2	Down
	3	Left
	4	Right

Returns:

None

Note: This moves the origin of the virtual screen within scroll margins. It only works if in Static or tracked mode, and has a similar effect to pressing the Alt-Cursor keys.

Fn 12H Screen refresh 2.1.4, 2.8, 3.7

Parameters:

AH	12H

Returns: None

Note: This service copies the contents of the Video RAM to the LCD controller, and is slightly faster than invoking Int 10H service 0.

Fn 15H Sound generation 2.1.5

Parameters:

AH	15H	
AL	Sub service:	
	0	Key-click
	1	Beep
	2	Alarm

Returns:

None

Fn 16H Melody tone generator 2.1.5

Parameters:

AH	16H	
CX	Length of tone in 10 mSecs intervals	
DL	Tone code (See below)	
30H	D#5	622.3 Hz
31H	E5	659.3 Hz
32H	F5	698.5 Hz
33H	F#5	740.0 Hz
34H	G5	784.0 Hz
35H	G#5	830.6 Hz
36H	A5	880.0 Hz
37H	A#5	932.3 Hz
38H	B5	987.8 Hz
39H	C6	1046.5 Hz
3AH	C#6	1108.7 Hz
29H	D6	1174.7 Hz
3BH	D#6	1244.5 Hz
3CH	E6	1318.5 Hz
3DH	F6	1396.9 Hz
0EH	F#6	1480.0 Hz
3EH	G6	1568.0 Hz
2CH	G#6	1661.2 Hz
3FH	A6	1760.0 Hz
04H	A#6	1864.7 Hz
05H	B6	1975.5 Hz
25H	C7	2093.0 Hz
2FH	C#7	2217.5 Hz
06H	D7	2349.3 Hz
07H	D#7	2489.0 Hz

Returns:

None

Fn 17H Dial number 2.1.5

Parameters:

AH	17H	
DS:SI	String of characters	
CX	Length of string	

Returns:

None

Note: String to be in ASCII. Valid characters are: 0 1 2 3 4 5 6 7 8 9 A B C D * #. Letters must be in upper case.

Fn 18H Mute states 2.1.5

Parameters:

AH	18H	
AL	00	Get mute state
	01	Set mute state
	02	Get key click state
	03	Set key click state
	04	Get bleep state
	05	Set bleep state
	06	Get alarm state
	07	Set alarm state
	08	Get DTMF duration
	09	Set DTMF duration
	If AL =	1, 3, 5, 7 or 9
	DL	0 Off (Muted)
		1 On

Returns:

	If AL =	0, 2, 4, 6 or 8
	DL	0 Off (Muted)
		1 On

Fn 19H Get Serial port parameters 2.7, 4.1, 3.5

Parameters:

AH	19H
DX	Serial port number

Returns:

If AH=0, Composite parameters in AL
If AH<>0, Error

Note: This service returns composite parameters identical to those used by Int 14H Service 0 (Initialize).

Fn 1AH Get Peripheral ID byte 2.7, 3.5

Parameters:
None
Returns:

AH	Peripheral ID byte
AL	0 if no peripheral installed

Note: This returns the peripheral ID code for the current terminating peripheral. (See Fn 1BH.)

Fn 1BH Set Peripheral ID byte 2.7, 3.5

Parameters:

AH	1BH
AL=0	Set Serial ID
AL=1	Set Parallel ID
DL	Current peripheral ID

Returns:
None

Note: There may be peripherals designed that contain circuitry that is similar to the Serial or Parallel peripherals. In order that these peripherals may use existing BIOS services they must identify themselves as being software compatible. DL should be set to the Peripheral ID code. (See Fn 1AH.)

Fn 1CH Preset/Return Peripheral data 2.7, 3.5, 4.1

Parameters:

AH	1CH
AL=0	Preset Data values
AL=1	Return Data values
BH	Table entry number
If AL=0	
BL	Data value
DX	IO address

Returns:

If AL=1	
BL	Data value
DX	IO address

Note: This service is used to preset peripheral IO data in a table associating an IO address with a data value. Service 0 will actually output the data to the specified IO locations. On Power-up, the table entries will be scanned for non-zero IO address values, and the associated data will be written out. This would typically be used to restore Interrupt numbers following Power-up. The first four table entries out of 10 max are reserved.

Fn 1EH Get/Set Clock tick speed 2.4, 3.8

Parameters:

AH 1EH
AL Subservice
- 0 Get speed
- 1 Set speed

If AL = 1
BX Clock tick speed
- 0 Tick every 128 seconds
- 1 Tick every second

Returns:

If AL = 0
BX Clock tick speed
- 0 Tick every 128 seconds
- 1 Tick every second

Note: 1 sec speed uses much more power.

Fn 1FH Get-key/NMI invoked screen refresh 2.8, 3.7, 3.8

Parameters:

AH	1FH
AL=0	Get refresh state
AL=1	Set refresh state
If AL=1	
DX	New state

Returns:

If AL=0	
DX	Current state
If AL=1	
DX	Old state

Note:

DH	Refresh on NMIs state
DL	Refresh on keys state
DH/DL=0	Revectoring disabled
DH/DL=1	Revectoring enabled

If bit 7 of the state is set, then the state is unchanged.

Fn 20H Disable revectoring of Int 9H 3.2

Parameters:

AH	20H
AL=0	Get revectoring of Int 9H state
AL=1	Set revectoring of Int 9H state
If AL=1	
DL=0	Disable revectoring
DL=1	Enable revectoring

Returns:

If AL=0	
DL=0	Revectoring disabled
DL=1	Revectoring enabled

Note: This is used to automatically revector Int 9H to the BIOS. This prevents applications software from setting up its own Int 9H. Note that the Portfolio keyboard IO address is not IBM compatible. This service is automatically invoked on a boot.

Fn 24H Get/Set ROM/CCM space state 2.2.4

Parameters:

AH	24H
AL=0	Get ROM state
AL=1	Set ROM state

If AL=1

DL	New ROM state
DH	New CCM state

Returns:

If AL=0

DL	Current ROM state
DH	Current CCM state

If AL=1

DL	Old ROM state
DH	Old CCM state

Note: ROM state in DL is as follows:

DL=0 Normal applications ROM
DL=1 CCM Drive A:
DL=2 CCM Drive B:
DL=3 Expansion ROM

CCM state in DH is as follows:

DH=0	CCM Drives Disable(d)
DH=1	CCM Drive A: Permanently enable(d)
DH=2	CCM Drive B: Permanently enable(d)

CF=0	No error
CF=1	Invalid option or error

Note: This service should be used with care, as it can swap either Memory cards or an extension ROM into the C000:0 to 0DFFF:F address range. This range is normally used by the internal applications ROM. Its primary use is to allow advanced users direct access to extension ROMs and Memory cards.

Fn 26H Get/Set Power control 2.1.10, 2.5, 3.8

Parameters:

AH	26H
AL=0	Get Power control state
AL=1	Set Power control state

If AL=1

DL	New state

Returns:

If AL=0

DL	Current state

If AL=1

DL	Old state

Note:

DL=0	Normal Power-down on low battery
DL=1	Prevent Power-down but display warning
DL=2	Prevent Power-down with no warning

This is used to prevent the Portfolio from powering down on a low battery. It is not recommended for use except for conditions in which a power down might be critical to an application or peripheral.

Fn 28H Get/Set Text/Keyboard language

Parameters:

AH	28H
AL=0	Get Languages
AL=1	Set Languages
AL=3	Language table pointers

If AL=1

DX	New languages

Returns:

If AL=0

DX	Current languages

If AL=1

DX	Old languages

If AL=3

ES:CX	Keyboard table pointer
ES:DX	Language table pointer

Note:

DH	Text language
DL	Keyboard language

Both DH and DL will be 0, 1 or 2, corresponding to the language in the ROM.

If bit 7 of the language/keyboard code is set, then it remains unchanged.

The tables consist of a count byte, followed by the language identification codes for the resident languages. These are as follows:

ENGLISH	0
FRENCH	1
GERMAN	2
SPANISH	3
ITALIAN	4
SWEDISH	5
DANISH	6

Fn 2CH Get BIOS version number

Parameters:
AH 2CH
Returns:
DS:BX Address of BIOS version number

Note: The version number consists of a Major and Minor version number, followed by a '$' terminator. A typical example is: '1.050$'

Fn 2DH Turn system off 2.1.10, 2.5, 3.8

Parameters:
AH 2DH
Returns:
None

Note: This is similar to typing OFF at the command line.

Fn 2EH Enable/Disable system status line

Parameters:
AH 2EH
AL=0H Disable status line
AL=1H Enable status line
DH Row number
DL Column number
Returns:
None

Note: This is similar to invoking the status line using the LOCK key.

Parameters:

AH	30H	
AL	0	Transmit block
	1	Receive block
	2	Open ports
	3	Close ports
	4	Wait 500mS
DS:DX	Start of Data buffer	

If AL= 0
CX Bytes to Send
If AL= 1
CX Maximum buffer size

Returns:

If AL=1
CX Bytes Received
DL Error Code

0	No error
1	Buffer size too small
2	Timeout on transmission
3	Checksum failure
4	Invalid sub-service
5	Peripheral not installed

Note: This is used by the File Transfer utility built into System Setup.

3.3.2 Disk services

The Portfolio Credit Card Memory (CCM)/Disk services are provided at the BIOS level by Int 13H.

There are six standard diskette sub-services, plus one special service. These are as below:

0H	Reset CCM/Disk system
1H	Get CCM/Disk status
2H	Read CCM/Disk sectors
3H	Write CCM/Disk sectors
4H	Verify CCM/Disk sectors
5H	Format CCM/Disk track
83H	Write CCM/Disk boot sector

Services 0 to 4 are similar to standard IBM PC BIOS disk services. They can access the three internally supported disk drives A, B and C (referred to as drives 0, 1 and 2 respectively).

Int 13H uses the BIOS Parameter Block (BPB) on the Boot sector (first sector) of the drive to determine the drive characteristics. During formatting, it is necessary to use a Format BPB, which is supported by service 83H. This service is used instead of service 5H to format the first track of a CCM/Disk.

The parameters to service 5 are unlike those on a normal PC as detailed below:

Int		13H Fn 5H
Parameters:		
	AH	5H
	DL	Drive number
	DH	Side/Head
	CH	Track number
Returns:		
	CF=1	Error code in AH

Note: Writes defined byte onto one track of CCM. Byte is specified in the Disk base table.

The Disk base table is similar to that used by an IBM PC. The table for both CCMs is pointed to by interrupt 1EH, and the table for the internal disk is pointed to by interrupt 41H. The format of both disk base tables is as below:

Offset 03H Bytes per sector code (0=80H, 1=100H, 2=200H)
Offset 0AH Format data bytes (Normally F6H)

During formatting using Interrupt 61H (see section 3.3.1), the CCM/Disk sector size is dynamically set according to the disk size. See below:

Disk size	Sector size
0 to <=32 Kbytes	80H/128 bytes per sector
>32 to <=64 Kbytes	100H/256 bytes per sector
>64 Kbytes	200H/512 bytes per sector

This ensures that a small disk size allows a reasonable number of sectors. Since Portfolio DOS allocates one sector per data cluster, this allows the same number of small data files on a 32K CCM as a 128K CCM.

There are various Int 61H services that provide extended disk services (see section 3.3.1.):

Int 61H Fn 7H	Format a CCM
Int 61H Fn 8H	Get the size of the Internal Disk
Int 61H Fn 9H	Format the Internal disk
Int 61H Fn 0BH	Determine if a valid CCM is present.

Note: A CCM may also contain a BIOS extension which does not affect the operation of the CCM, but can modify the Operating system or Power-down/Power-up sequence. (See section 3.5.)

3.4 Differences Between Portfolio DOS and MS-DOS

For the purposes of this manual, Portfolio DOS is defined as the program which communicates between the Command processor or User application, and the BIOS. It does NOT include the Command processor. (See recommended books in section 1 for more information on the standard MS/PC-DOS.)

There are a few differences between Portfolio DOS and MS-DOS. These are mainly enhancements to Portfolio DOS 2.11 to make it more DOS 3.XX compatible:

Int 21H Fn 37H Get/Set Country
: Portfolio DOS is DOS 3.XX compatible

Int 21H Fn 4BH Execute program
: As well as providing standard EXEC services it also allows a program to be RUN directly off a CCM (section 3.6.3).

Int 28H Keyboard busy
: Not supported. This would normally be called during console IO polling, however Portfolio DOS does not poll the console, but actually waits for a key using Int 16H Fn 0H. (See section 3.8 on power management.)

Int 2AH Internal MS-DOS function not fully supported

3.5 Device Drivers and Peripheral Software

3.5.1 Device Drivers

Device drivers are used by DOS to communicate with the BIOS. They provide a standard interface which isolates the DOS from the device specific BIOS. The Portfolio has the following resident device drivers in ROM:

CON, CLOCK$, PRN, LPT1, AUX, COM1 and Disk driver

CON	performs all Console IO
PRN/LPT1	perform all Parallel (Printer) IO
AUX/COM1	perform all Serial IO
CLOCK$	special driver to access the BIOS Clock

These are all character devices that process strings of characters one character at a time. They are all identified by their names.

The Disk device driver is a Block device which requires all IO to be done in blocks. It addresses all the normal Portfolio disk drives (A, B and C). It has no name.

It is possible to replace these resident device drivers (and add new ones) by the use of installable device drivers. These may be loaded by DOS using the 'DEVICE=' command in CONFIG.SYS. If a character device is loaded that has the same name as one of the above device drivers, then it replaces it. This mechanism is used by programs such as ANSI.SYS which is actually a CONsole device driver with added features.

If a Block device driver is added, it supplements the existing Disk device driver. An example of this is the Virtual disk driver VDISK.SYS, which would add drive D:.

The structure of an installable device driver is compatible with any MS-DOS 2.11 device driver.

3.5.2 Peripheral Design

There is a special design issue associated with Portfolio peripherals, due to the Portfolio auto power-down power conservation feature. This means that most peripheral devices will need to be re-initialized on power-up. (See Appendix C for more information.)

There are two methods provided to fulfill this requirement:

1) Int 61H Fn 1CH
This service stores a list of IO addresses and associated data values, which will be output on a power-up. If all initialization specific IO writes are made via this service, then they will automatically be repeated on all power-up sequences.

A typical use for this service might be to restore an interrupt number in an interrupt driven serial peripheral.

2) Use a ROM extension. This would generally be required when the sequence of operations during power-up could not be supported by the Int 61H service. This will require the peripheral to contain an extension ROM. (See 3.5.3.)

There are two exceptions to the above. The serial port parameters are read during the power-down sequence and correctly re-programmed on the subsequent power-up. The Parallel port is also initialized on Power up.

Each peripheral is identified to the Portfolio by its Peripheral ID code (PID) (see section 2.7). This is actually a hardware IO location on the peripheral which may be read using Int 61H Fn 1AH.

The other software issue associated with custom peripheral design concerns the Serial or Parallel peripherals. If the custom peripheral wants to use existing BIOS services then they must identify themselves as being hardware compatible:

Int 61H Fn 1BH configures the BIOS to recognize a peripheral to be Serial or Parallel compatible.

3.5.3 ROM Extensions

ROM extensions are sections of code that can be executed at various stages during the BOOT sequence, and during Power Up and Power Down. They may be on a Credit Card Memory (CCM) or on a extension ROM on a peripheral. A typical use of such an extension is to modify the operating system or initialize custom peripherals.

There are three main types of extensions: A Specific BIOS extension, a Specific DOS extension, and Common extensions:

* The Specific BIOS extension is invoked after BIOS initialization.
* The Specific DOS extension is invoked after DOS initialization.
* The Common extension is invoked before and after both BIOS and DOS initialization, before Command processor initialization and during Power-Down and Power-Up.

The ROM extensions are searched for on Drive A, then the extension ROM and then Drive B. If a valid extension is found and executed, then the search for that particular type of extension is terminated.

The format of a ROM/CCM extension is as follows:

```
Offset    Size
00H       dw      ?               ;Identification code
02H       db      ?               ;Number of 512 byte
                                  ;blocks(unused)
03H       db      5 dup (?)       ;Specific BIOS/DOS exten.
40H       db      'DIP ROM!!'     ;OEM user text
50H       db      5 dup (?)       ;Pre-bios jmp vector
55H       db      5 dup (?)       ;Bios-ext jmp vector
5AH       db      5 dup (?)       ;Pre-dos jmp vector
5fH       db      5 dup (?)       ;Dos-ext jmp vector
64H       db      5 dup (?)       ;Post-dos jmp vector
69H       db      5 dup (?)       ;Power-Down jmp vector
6eH       db      5 dup (?)       ;Power-Up jmp vector
```

The extension vectors occupy the first 128 bytes of the CCM/ROM. The vectors are positioned so as to allow a valid BIOS Parameter Block (BPB) on a CCM so that it can be used both as an extension CCM and as normal. The Identification code at Offset 0 determines the main extension type as below:

```
AA55H     ;Specific BIOS extension
55AAH     ;Specific DOS extension
5555H     ;Common extensions
```

Thus, if the word at Offset 0 is AA55H, then after BIOS initialization a FAR CALL will be made to Offset 3. The 5 bytes following this offset allow for a short/normal/far jump to the extension code. If the word was 55AAH, the call would be made after DOS initialization. If the word is 5555H, then all the common extensions would be called at the appropriate times.

Note: All the jump vectors must be set up to a suitable return when using a common extension, even if they are not used.

All ROM extensions must preserve the processor registers. Extreme care must be taken when using extensions, especially those which are invoked half way through the boot sequence, as these may adversely affect the operation of the Portfolio. The Pre-BIOS extension is called almost immediately on jumping from the Reset vector, and so has no stack set up. It must return via a FAR JUMP to 0FFFE:0H. All the other extensions must return via a FAR RET. It is recommended that the Post-DOS extension is used in preference to those preceding it.

The OEM user text field at Offset 40H is to allow an OEM to identify the ROM.

See Appendix C for examples of using an Extension ROM.

3.6 Memory Cards

3.6.1 Format

Each credit card memory (CCM) must be formatted before use, this program creates a format analogous with a standard floppy disk format.

All formatted memory cards contain only 1 sector per cluster as opposed to the 2 or more found in larger systems. (See section 3.3.2 for more details.)

The Atari Portfolio BIOS has been written to handle future paged Credit Card Memories (CCM). The BIOS assumes that the page register, is one byte located at offset 10 (0AH) within the Boot sector (First sector) of the Memory card. For this reason DO NOT use this memory location in programs.

3.6.2 Autoboot Mechanism

The Portfolio has the ability to invoke AUTOEXEC.BAT from drives other than drive C:.

If a memory card is in drive A: or B: and AUTOEXEC.BAT file exists, it is executed in preference to autoexec on C:. B: will have priority over A: if an AUTOEXEC.BAT exists on both A: and B:.

If it is required that the AUTOEXEC.BAT on drive C: is always executed, terminate the batch file on drive A: with the command:

```
C:AUTOEXEC
```

The CONFIG.SYS file is always loaded from drive C: and cannot be overridden.

3.6.3 Run

A RUN file is a specially written program that can be directly executed from a Credit Card Memory (CCM) without having to be loaded into the Transient Program Area (TPA). An obvious advantage of this method of execution is that it minimizes system memory usage.

A RUN file can be executed from the Command processor by typing RUN <filename>, or by invoking Int 21H Fn 4BH at the DOS level as for a normal program, but with AL set to 80H.

There are several requirements for the programs which can use the RUN command:

* The program needs to be specially written to be used with the RUN command.
* The program needs to be on drive A: or drive B: and it needs to occupy consecutive clusters on the disk. This situation cannot be guaranteed if a file is simply copied to the drive.
* The file needs to have a .RUN extension.

Writing .RUN Programs

Almost all standard programs assume either that their data is in system RAM, or that they can store data in their code segment. Although a .RUN file is similar to a .COM file, care should be taken when dealing with data.

The initialization code of the .RUN program has to perform the following (these points are illustrated in Appendix A):

* Reduce the system memory usage down to the minimum requirement. There must be at least 10h paragraphs, i.e. the size of the PSP.
* Allocate data and stack using DOS interrupt 21h fn 48h and set ss:sp to point to this block.
* Copy all initialized variable data from the memory card to the allocated data block in system RAM.
* The program can then perform most functions it wishes to, including any DOS calls. The program MUST terminate with DOS interrupt 21h function 4Ch "terminate process".
* The program does not need to copy the non-variable (constant) data from the ROM card into RAM before using it. This data can be used directly from the card. This means prompts or text messages need not take up any RAM.

The built-in applications can be invoked while a program is being RUN using the hot keys as usual.

If during execution of a RUN file the Credit Card Memory (CCM) is removed from the drive, the next instruction to be executed on the card will be interrupted by an error handler. This displays the error message: 'ERROR: Card Access' and terminates the process. A RUN file may not EXEC another file from within itself.

Appendix A provides an example of a RUN program which illustrates how to program a typical RUN program.

3.7 Screen Handling

The Portfolio BIOS supports two main Video modes, Text and Graphics. The BIOS Video Interrupt 10H may be used to set the screen mode:

Mode	Type	PC Resolution	Portfolio
Mode 7	Text	(80,25)	<80,25>
Mode 8	Graphics	(160,200)	<240,64>
Mode 9	Graphics	(320,200)	<240,64>
Mode A	Graphics	(640,200)	<240,64>

As can been seen, the Portfolio interpretations of these modes is fairly simple.

The Text modes are actually viewed using a 40x8 window. There are 3 sub-modes: Static PC, 40x8 and Tracked.

Static PC mode is where the physical screen area acts as a window onto the larger 80x25 text screen. The window may be moved using the Alt-Cursor keys, or Int 61H Fn 11H.

40x8 mode actually sets the logical screen size to 40 columns by 8 rows. This mode is intended for use by software written specifically for the Portfolio, such as the the Command processor and the Internal applications.

Tracked mode is similar to static mode, except that the Screen window positions itself at the cursor.

This mode can be set using Int 61H Fn 0EH.

The Video RAM (VRAM) for the text screen is at segment 0B000H therefore it is possible to write directly to the Video RAM, but any screen refreshing must be invoked by the application. There is only one text page.

There are other Int 61H Video services:

Int 61H Fn 0DH	Get the logical and physical screen sizes
Int 61H Fn 0FH	Set cursor size
Int 61H Fn 10H	Set the virtual screen window origin on the 80 by 25 screen
Int 61H Fn 12H	Force a screen refresh

In Graphics mode, the Graphics screen has a 240 by 64 pixel resolution and can be written to or read from using BIOS pixel read/write Int 10H Fn 0CH or Fn 0DH respectively. The Atari Portfolio has three cursor modes: Block, Underline and Off. If the cursor size is set in the BIOS then either Block or underline mode will be set up.

3.8 Power Management

The Portfolio is designed to minimize power consumption and hence maximize battery life. This is reflected in the hardware design, but is enhanced by various software features.

The main power wasting operation in most computers is waiting for user entry at the keyboard. Once the Portfolio Keyboard BIOS Getkey ready service (Int 16H Fn 0) has detected keyboard inactivity, it will start to decrement a timeout counter. On timeout, the Portfolio will enter its power-down sequence. Once powered-down, any hardware interrupt will initiate a power-up sequence.

This timeout is dependent on whether the machine is set to fast or slow timer ticks (Int 61H Fn 1FH), but is always between 128 and 256 seconds.

It is important that all keyboard input is done via a DOS or BIOS keyboard service that waits for a key press. Polling the keyboard continuously will quickly wear out the batteries. This will be obvious as the Portfolio will never power down.

Another power wasting operation is refreshing the LCD controller from the Video RAM. If an application writes directly to the Video RAM, then it must be refreshed at appropriate intervals using Int 61H Fn 12H. It is possible to force a screen refresh on a keypress or on a timer-tick using Int 61H Fn 1FH. Many applications which run on the Portfolio, but are designed for the IBM PC require this refresh on keys as they assume automatic screen refreshing. Refresh on timer ticks is dependent on the tick speed. This can be set using Int 61H Fn 1EH.

Note: The timer tick (Int 8H and Int 1CH) is not the same as the IBM PC timer tick which occurs 18.2 times a second. It is either generated once every 128 seconds, or once per second.

As a general rule, an application should avoid refreshing the screen except where necessary. With refresh on both keys and timer ticks, and with timer ticks set to fast (i.e. 1 tick per second), the processor spends a lot of its time refreshing the screen.

Generating sounds using Sound generation, Melody tone generation or Tone dialling (Int 61H Fn 15H, 16H or 17H respectively) can draw a high current from the Portfolio batteries.

The Alarm will timeout after about 15 seconds to prevent the batteries from being overly strained.

All peripherals will add to the power consumption, unless they have their own power source. It is recommended that an external power supply is used wherever practical when using peripherals.

During Disk access, Tone generation, Timer tick and on each press of the return key, the Portfolio checks to see if its batteries are running low. If a low battery is detected, the Portfolio automatically powers down after displaying a low battery message. On power up, it will display the same message to indicate to a user the reason why it powered down.

It is possible to prevent the Portfolio from powering down using Int 61H Fn 26H. This service must ONLY be used if absolutely necessary, because this may force the Portfolio to operate outside its electrical specification with possible damage to the hardware.

3.9 Special File Formats

3.9.1 Diary saved file format

The built in diary saves its data in a standard ASCII file format. To eliminate different date formats for different countries, the Diary stores the information in English format.

Certain information, such as repeat entries and alarm information is saved along with the diary entries.

The following is an example diary file:

```
6/07/89
14:14 Technical reference
20:00 Go home
```

If an entry is a repeating entry, the time is preceded by a code letter indicating the repeat period. The following table lists the code letters and their associated repeat periods:

d	Daily
w	Weekly
n	Non weekend, i.e. Mon-Fri
m	Monthly
y	Yearly

If an alarm is associated with an entry, the '@' symbol is placed on the line before the time.

If an entry has an alarm and it is a repeating entry, the repeat symbol preceeds the alarm symbol.

For example:

```
      6/07/89
@     20:00 Go home
      7/07/89
d     14:48 Hello there!
w@    18:10 Goto tennis
```

The first entry is a non repeating entry with an alarm. The second is a daily repeat and the third is a weekly repeat which will sound the Portfolio's alarm.

The diary sorts the entries chronologically when it loads any given file.

3.9.2 System File Formats

There are three files used by the system which obey a standard file format. These are:

Clipboard (C:\SYSTEM\CLIPBORD.DAT)
Undelete (C:\SYSTEM\UNDELETE.DAT)
Permanent data (C:\SYSTEM\PERMDATA.DAT)

All three files are loaded into RAM when the applications are invoked. Operations affecting any of the information stored in these files only change the RAM copy. All three files are written out when the user quits all the applications, i.e. presses <ESC> at the top level menu.

3.9.2.1 CLIPBORD.DAT

This is the file which the applications use as the clipboard. It is a single block of data ending with a zero byte (00h). Carriage returns are stored as 0Dh without the trailing 0Ah Line feed. The maximum size of the clipboard is 8K characters. This must include the 0 terminator.

If the file does not contain a 00h termination byte, then the file is not loaded into the clipboard. Similarly, if the file is greater than the maximum number of allowed characters, it isn't loaded. In both cases when the file is not loaded, it will be overwritten with a fresh file upon exit from the application.

If the clipboard has the normal text format of 0Dh,0Ah the file will only load correctly into the Editor and the Diary.

3.9.2.2 UNDELETE.DAT

The undelete file is used to store all the characters or blocks of data deleted from all the applications. It is made up of a number of "blocks" of data. Each block represents a group of characters deleted with one command. The format of a block is as follows:

<DATA> <00h> <DIR>

The <DATA> is the character or characters which are deleted. If a block of data is deleted containing carriage returns, these are stored as <0Dh>, not <0Dh><0Ah>.

The 00h byte is used to determine the length of the deleted block.

The <DIR> is a one byte code indicating in which direction the deletion was made. If the data was deleted to the left, i.e. using the BACKSPACE key, then this byte will contain <00h>. If the data was deleted from the right, i.e. using <DEL>, this byte will be <01h>.

The maximum number of characters which the undeletion file can contain is 2000. If the file contains more than this number of bytes, then it will be ignored and replaced with a new file upon exiting from the application.

If the UNDELETE.DAT file doesn't have the correct format, the effects are unpredictable. It is likely the data in the file will be inaccessable.

3.9.2.3 PERMDATA.DAT

The format of the system data file is as follows:

Byte(s) in Hex	Function
0	Non zero: undelete buffer enabled.
1	Non zero: undelete buf is saved on exit
2	Non zero: clipboard is enabled.
3	Non zero: clipboard is saved on exit
4..6	Reserved.
	Worksheet:
7	Non zero: frame on upon entry.
8..56	Drive/Path/Name of last used file.
57	0: Autoload last spreadsheet, 0ffh don't
58..5b	3 character 0 terminated currency string.
5c	Initial decimal point, '.'(2eh) or ','(2dh)
5d	Printer default paper width.
5e..60	Reserved.
	Diary:
61	Non zero: frame on upon entry.
62..b0	Drive/Path/Name of last used file.
b1..b4	Reserved.
	Editor:
b5	Non zero: frame on upon entry.
b6..104	Drive/Path/Name of last used file.
105..106	Top line on screen 0 is first line in file
107..108	Current cursor line no, 0 is first line.
109..10a	Cursor: number of bytes into current line.
10b	Right margin.
10c	Non zero: word wrap on.
b0d..10e	Reserved.

	Address book:		
10f	Non zero: frame on upon entry.		
110..15e	Drive/Path/Name of last used file.		
15f..168	Dial prefix in ASCII, zero terminated.		
169..16f	Reserved.		
	Calculator:		
170	Non zero: frame on upon entry.		
171..1bf	Reserved.		
1c0	M1 sign: Bit 7 set for negative number.		
1c1..1c2	M1 exp: signed word, 0:1.000<=mant<2.000		
1c3..1ca	M1 mantissa.		
1cb..1d5	Memory 2.		
1d6..1e0	Memory 3.		
1e1..1eb	Memory 4.		
1ec..1f6	Memory 5.		
1f7	Format: 0-General, 1-Fixed, 2-Sci., 3-Eng.		
1f8	Number of decimal places		
1f9	Separators: 0 - none, non-zero-separators		
1fa	Decimal point: 0 - '.', non-zero ','		
	Setup:		
1fb	Reserved.		
1fc..24a	Drive/Path/Name of printer file destination		
24b..24e	Reserved.		
24f	Printer dest, 0:Parallel, 1:Serial, 2:file.		
250	Lines per printer page.		
251	Printer setup code length.		
252..2d1	Setup codes, ASCII chars, i.e. ESC=1Bh.		
2d2	End of line code	0	<CR>
		1	<CR><LF>
		2	<CR><LF><LF>
2d3	Number of top paper margin lines.		
2d4	Bottom paper margin lines.		
2d5	Left paper margin character indent.		
2d6	Baud rate	0	110
		1	150
		2	300
		3	600
		4	1200
		5	2400
		6	4800
		7	9600
2d7	Parity	0	None
		1	Odd
		3	Even
2d8	Data bits	2	7 bits.
		3	8 bits.
2d9	Stop bits	0	1 stop bit.
		1	2 stop bits.

The PERMDATA.DAT file currently contains 730 bytes.

3.10 IBM PC Development system

The Portfolio contains a few system specific extended BIOS functions which are accessed using interrupt 61H. If you want to emulate these functions when developing programs on a standard IBM PC then this is possible by running the program I61.EXE on a PC. This program will remain Terminate and Stay Resident (TSR) and hence enable you easily develop custom programs. To ensure upward compatibility of your programs, if you plan to use any I61 functions then make sure that you use I61 Fn 0H first. (See section 3.3.1.)

Int 61H Services supported by IBM hosted version:

Fn No	Function description	Supported
0H	Service Initialization	✓
7H	Format Credit Card Memory	x
8H	Get size of Internal disk	✓
9H	Format Internal disk	x
BH	Determine if CCM present	x
DH	Get/Set Screen size	✓
EH	Get/Set Screen mode	✓
FH	Get/Set Cursor mode	✓
10H	Get/Set virtual screen position	✓
11H	Move virtual screen position	x
12H	Screen refresh	x
15H	Sound generation	✓
16H	Melody tone	✓
17H	Dial number	✓
18H	Mute states	✓
19H	Get Serial port parameters	✓
1AH	Get Peripheral ID byte	x
1BH	Set Peripheral ID byte	x
1CH	Preset Peripheral IO data	x
1EH	Get/Set Clock tick speed	✓
1FH	Get-key/Tick Screen refresh	½
20H	Disable revectoring of Int 9H	½
24H	Get/Set ROM space state	x
26H	Get/Set Power State	x
28H	Get/Set Language	✓
2CH	Get BIOS version number	✓
2DH	Turn system off	x
2EH	Enable/Disable status line	✓
30H	File transfer via smart cable	✓

Key: ✓ Service supported
x Service not supported
½ Service partly supported

4 PERIPHERALS

4.1 Portfolio Serial Communications

4.1.1 Hardware Specification

Standard:	EIA RS232C compatible
Line Voltages:	+/- 9V
Current Loop:	Not Supported
Connector:	9 Pin D-Shell Plug (AT compatible)

Connector Pin out:

Pin	Name	
1	CD	Carrier Detect
2	RD	Receive Data
3	TD	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Signal Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicator

Interface IC:	82C50A
Base Address of 82c50:	Stored at Memory Location 400h
Interrupt Support:	Yes (see below)
Divisor Clock:	1.8432 MHz

4.1.2 IO Registers

Since the same computer interface (with the exception of interrupt handling) is used on the Portfolio as on the IBM PC/AT, the IO registers have the same function. The base address for the serial port may be found by reading memory location 400h in the BIOS data area. If the value at this address is XXXXh, then the IO registers are thus:

IO Address		Register of 82c50A	
XXXX+0	R	RBR	Receiver Buffer Register
	W	THR	Transmitter Buffer Register
XXXX+1	R/W	IER	Interrupt Enable Register
XXXX+2	R/W	IIR	Interrupt Identification
XXXX+3	R/W	LCR	Line Control Register
XXXX+4	R/W	MCR	Modem Control Register
XXXX+5	R/W	LSR	Line Status Register
XXXX+6	R/W	MSR	Modem Status Register
XXXX+7	R/W	SCR	Scratch Register

4.1.3 Interrupt Support

Since the Portfolio does not contain an 8259 compatible Peripheral Interrupt Controller, interrupts are handled in a different way than on an IBM PC/AT.

The serial port has register called the Serial Interrupt Vector Register (SIVR). An eight bit number can be written to this register. This number is the interrupt number that is to be used with the serial port. For example, writing 10 to SIVR will cause a call to the double word pointer held at memory address 10*4.

SIVR is at I/O location 807Fh and is write only. It should be set up before 82c50A interrupts are enabled.

When an interrupt is generated by the 82c50A, it is passed on to the CPU. If no other interrupts are pending then the CPU will read the contents of SIVR and service that interrupt number.

Interrupts are acknowledged by accessing the 82c50A and reading IIR. This will allow the operation required to service and acknowledge the interrupt to be determined.

4.1.4 Other Useful Information

To determine whether a serial port is installed, it is recommended that use is made of BIOS Interrupt 11h - Get Equipment List.

Since the Portfolio will attempt to power down while waiting for a key stroke (INT 16h service 00h), it is recommended that terminal emulation software polls the keyboard until a key is waiting in the buffer (INT 16h service 01h).

To set up SIVR it is recommended that INT 61h service 1Ch is used:

```
AH = 1Ch
AL = 0              ;set up IO address
BH = 5              ;IO table entry 5
BL = Byte to write
DX = IO address
```

Use of this function will ensure that SIVR is always set up correctly (unless table entry 5 is reused for a different address).

In order to maintain future compatibility it is recommended that on exit from the program, the table entry used above have its address set to zero. This should be followed by a write of 2Ah to I/O 807Fh.

4.2 Smart Parallel Interface
File Transfer Protocol Description

The IBM PC and many compatibles have uni-directional centronics parallel ports. In order to allow an inexpensive but useful peripheral it was decided that the Portfolio parallel centronics port would allow programs to be sent to and from IBM PCs as well as to a printer. This is accomplished by using a synchronous serial transfer protocol. Status lines on the IBM PC which can be accessed through the BIOS are used as inputs on the IBM. The Portfolio parallel port is fully bi-directional.

The file transfer BIOS should be used with the following considerations (see section 3.3.1):

* Before sending or receiving the ports should be opened.

* Sending a block expects the other end to be receiving a block, and visa versa.

* A timeout will occur if there is no answer within 500mS.

* Sending a block will automatically transfer the length of the block. The receiver will return an error if the buffer is too small.

* On any failure, wait 500mS (to allow the other end to timeout) and attempt to re-transmit/receive the block.

* A error at one end will normally cause an error at the other, so block order should not be lost.

* A checksum will be sent with each block to provide simple error detection.

* At the end of the transfer the ports should be closed.

4.3 IBM PC Card Drive

The IBM PC card drive consists of an expansion bus card and plastic box with a cable. The expansion card can be used in a IBM PC/AT or compatible. The cable is used to connect the card to the plastic box. There is a slot in the front of the box that allows the insertion of a Credit-card memory, as used on the Portfolio.

By running the appropriate block device driver software, the card drive can access the card in the same way as a normal disk.

The card uses a block of four I/O locations. These are located at a start address indicated by optional links on the board. When these are changed from the default setting, the device drive driver must be told of the change in the CONFIG.SYS file.

4.4 EPROM Writer Adaptor Boards

PROM programming adaptors are available which allow PROM (OTP) memory cards to be programmed using a standard PROM programmer. The adaptors convert the PROM card to the same outline as a standard DIL PROM. Use model HPC-501 to program 512 KBit cards and model HPC-502 for 1 Mbit cards.

When programming the PROM card the PROM programmer should be set up as a Fujitsu PROM. If however, Fujitsu settings are not available, some of the other 12.5V PROM programming specifications will also work. The ideal programming specification is:

VPP	12.5V	
64 KByte	use 27C512	(Ideally Fujitsu CMOS)
128 KByte	use 27C1001	(Ideally NEC CMOS)

Once the correct ROM type has been chosen use the following procedure to make a copy of a RAM card:

i) Select a PROM card with the same capacity as the RAM card and use the correct adaptor for the card capacity

ii) Place adaptor in PROM socket of the programmer, ensure that it is inserted the correct way.

iii) Place RAM card in to adaptor and LOAD the contents in to the programmer using the relevant option.

iv) Place PROM card in to the adaptor and program as for a normal PROM chip.

WARNING:

1) DO NOT ATTEMPT TO PROGRAM THE RAM CARD

2) Some PROM programmers do not like the power being turned ON and OFF so remove the cards before switching ON or OFF.

APPENDIX A: EXAMPLE .RUN PROGRAM

Section 3.6 highlights the main design issues to take into account when creating a .RUN program.

Included on the Portfolio Emulator Disk are the following files:

RNRN.ASM	Assembler .RUN program
MAIN.C	Example C .RUN Program
RU_C.ASM	C Header
BUILD.BAT	Creates program using Turbo C tools

The above files illustrate two programs which use the Atari Portfolio .RUN function.

RNRN is an assembler program which prints out the original calling parameters and then three numbers.

MAIN is a program which illustrates a .RUN program written in C. RU_C.ASM is the C Header necessary for Borland's Turbo C compiler and BUILD.BAT illustrates how to create the program using Turbo C tools.

Notes on interfacing "C" files to the RUN command

For "C" files, several more segments need to be declared to ensure "C" gets the data and code in the correct positions in the .RUN file. The .COM file is converted into a .RUN file by renaming.

RU_C.ASM is the header which can be used to interface to a "C" program. If "C" source files are being used without any provided "C" libraries then the file RU_C can be used as the header directly. If library code is required the header will need to be enhanced to perform the necessary library initialization.

The RU_C.ASM header works with Turbo C and can be used as a guide to modifying other "C" headers used by different "C" compiler libraries.

Most "C" headers supplied with "C" compilers can be assembled for different memory models. The example code in RU_C.ASM needs to be placed in the "C" startup header and assembled for the SMALL model.

Run Files Greater than 64K

To build a .RUN file with a code size greater than 64K, it is necessary to have more than one code segment. One way to achieve this is to build the program

using the MEDIUM memory model. In this way, the code size is only limited to the available space on a CCM (up to 128K)

Unlike an .EXE file, which has fixups resolved at run time, a .RUN file must have the fixups resolved before the program is committed to a ROM card. Therefore it is necessary to resolve the fixups based upon an absolute memory address for the file, and it must be known in advance where the file will reside on the card. If the program is the first file on the card, its position will be calculated as follows:

Fixup Address (in paragraphs) =

C000H + (Boot sectors + FAT sectors + Root Dir sectors) * (sector size in paras)

The number of sectors used can be found by using a disk utility program (such as Nortons Utilities).

Example:

For a 128K card formatted with 512 bytes per sector, 1 sector for the Boot Record, 1 sector for the FAT, and 8 sectors for the Root Dir, the address (in paragraphs) of the first file on the card will be C140H.

This value should then be used for the fixup segment address, before the program is copied to the ROM card.

Due to the mechanism used by the operating system to execute .RUN files, the file must have an apparent size less than 64K. Therefore, after the program has been copied to the card, the file size entry in the Root Dir must be set to a value less than 64K.

Since data fixups must be resolved at run time, it is not possible to have more than 64K of data. This means that the HUGE memory model cannot be used.

```
TITLE   RNRN.ASM

comment *
        (c) Copyright DIP, 1989

        Example .RUN program

        *

DGROUP  group   _text,_data,_cdata

STACKSIZE       equ     400             ; byte in stack.

; code segment.
_text   segment public byte 'CODE'
        assume  cs:_text,ds:_data

        org     0                       ; IP is 0 on entry.

;*************************************************************************
; rnrn_main
;
; RUN command test routine.
; On entry, DS, SS and ES all point to the PSP in RAM.
; CS is a ptr into the credit card, so may actually be in ROM!
; When this routine is executed, the whole of RAM is allocated to the
; process.
;
; Parameters:
;       None
; Returns:
;       None
;*************************************************************************

rnrn_main       proc    near

        mov     bx,10h                  ; 10h paras to keep the PSP.
        mov     ah,04ah                 ; modify memory.
        int     21h
        jc      rnrn_err                ; error reducing memory.

        mov     bx,OFFSET rnrn_uend     ; alloc for initialised data,
        sub     bx,OFFSET rnrn_dstart   ; uninitialised data and a
        add     bx,0fh                          ; stack.
        mov     cl,4
        shr     bx,cl                   ; calc paras in init data area.
        add     bx,STACKSIZE/16         ; add in paras in stack.
        mov     ah,48h                  ; allocate memory.
        int     21h                     ; allocate stack and data.
        jc      rnrn_err                ; no memory.

        mov     ss,ax                   ; set stack to point to RAM.
        mov     sp,OFFSET rnrn_uend+STACKSIZE

        push    es                      ; preserve PSP pointer.

        mov     cx,OFFSET rnrn_dend
        sub     cx,OFFSET rnrn_dstart   ; bytes in initialised data.

        mov     si,OFFSET DGROUP:rnrn_dstart    ; copy from here.
        push    cs
        pop     ds                      ; source is on memory card.
        xor     di,di
        mov     es,ax                   ; target is allocated RAM.
        cld

        rep     movsb                   ; copy init data from card to RAM.

        pop     es                      ; restore PSP ptr.

        mov     si,5dh
        mov     cx,11
```

```
rnrn_fcb1:
        mov     dl,es:[si]              ; get char from FCB built into PSP.
        inc     si
        mov     ah,2
        int     21h                     ; print name of first parsed FCB.
        loop    rnrn_fcb1

        mov     si,6dh
        mov     cx,11

rnrn_fcb2:
        mov     dl,es:[si]
        inc     si
        mov     ah,2
        int     21h                     ; print name of second parsed FCB.
        loop    rnrn_fcb2

        push    ss
        pop     ds                      ; DS is ptr to data in RAM.

        mov     al,_rnrn_val            ; get initialised data.
        call    rnrn_disp               ; display the value.

        inc     _rnrn_val

        mov     al,_rnrn_val            ; get changed data.
        call    rnrn_disp               ; display the value.

        mov     rnrn_unin,44            ; set a piece of uninitialised data.
        mov     al,rnrn_unin            ; get uninitialised data.
        call    rnrn_disp               ; display the value.

        xor     al,al                   ; return errorlevel of 0.
        jmp     short rnrn_end

rnrn_err:
        mov     dx,OFFSET DGROUP:rnrn_mem
        push    cs
        pop     ds                      ; write directly from ROM card!
        mov     ah,9                    ; write string.
        int     21h                     ; tell user there was memory error.
        mov     al,1                    ; terminate with error code of 1.

rnrn_end:
        push    ax                      ; save errorlevel code in al.

        mov     ah,1                    ; wait for a key.
        int     21h

        pop     ax                      ; get errorlevel code back.
        mov     ah,4ch                  ; terminate process.
        int     21h

rnrn_main       endp

;*************************************************************************
; rnrn_disp
;
; Display the value in AL with a trailing space.
;
; Parameters:
;       AL      Value to print, less than 100.
; Returns:
;       None
;*************************************************************************
rnrn_disp       proc    near
        aam                             ; convert to two numbers.
        add     ax,3030h                ; convert to ASCII digits '0'..'9'.
        push    ax                      ; save to print 2nd char.

        mov     dl,ah
        mov     ah,2                    ; print 1st digit.
        int     21h
```

```
        pop     dx
        mov     ah,2                    ; print 2nd digit.
        int     21h

        mov     dl,' '
        mov     ah,2                    ; print a space.
        int     21h

        ret
rnrn_disp       endp

_text   ends


; initialised and uninitialised RAM data.
; this is para since the segment will start at zero when it is copied
; over into RAM.
_data   segment public para 'data'
rnrn_dstart     label   byte

        public  _rnrn_val
_rnrn_val       db      42

rnrn_dend       label   byte
rnrn_ustart     label   byte            ; uninitialised data start.

rnrn_unin       db      ?

rnrn_uend       label   byte            ; uninitialised data end.
; the stack is added on here, after initialised and uninitialised data.
_data   ends


_cdata  segment public byte 'data'

; initialised data which doesn't get transferred to RAM.
rnrn_mem        db      "Out of memory$"

_cdata  ends
        end     rnrn_main
```

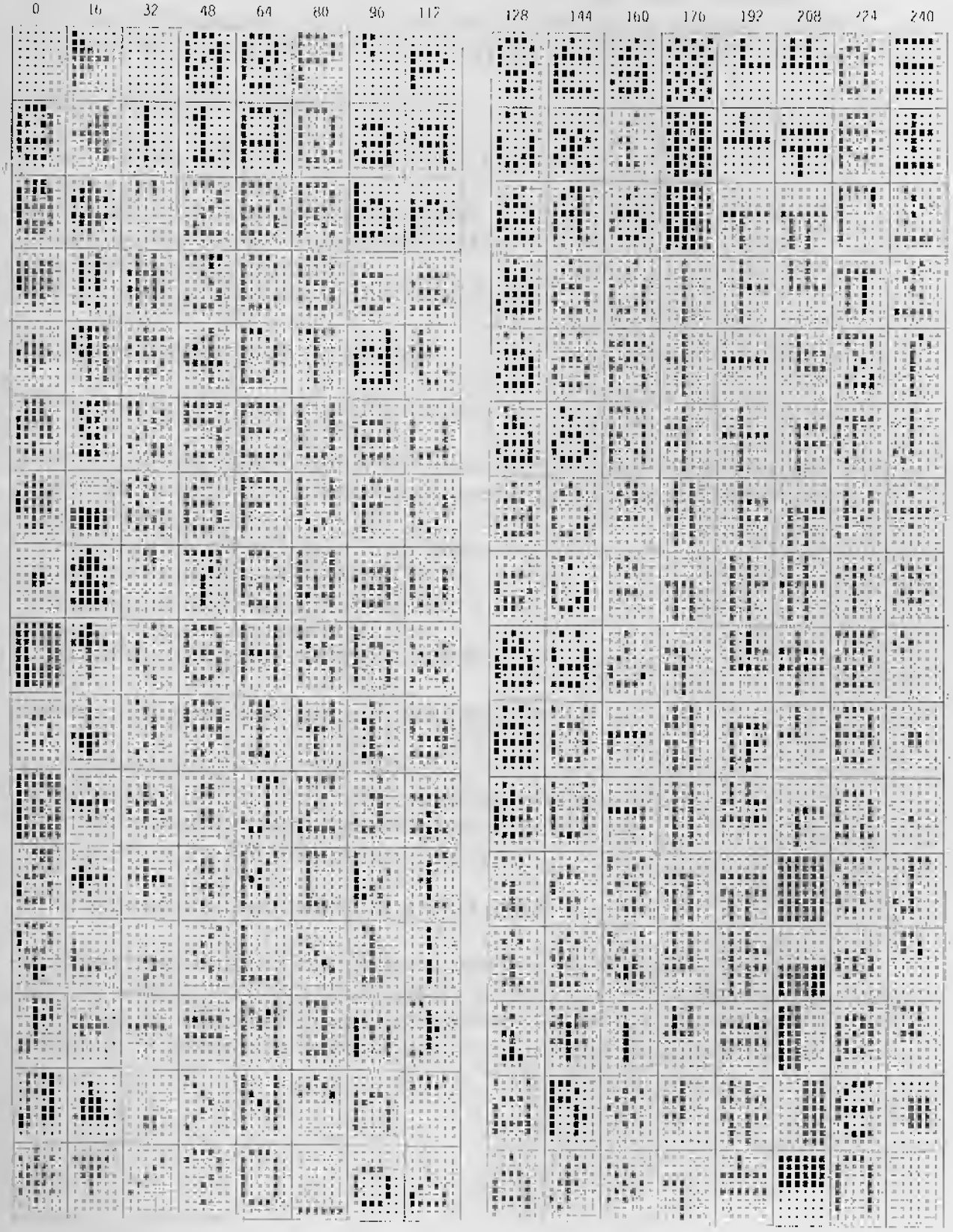
```
/*      MAIN.C

        Copyright DIP Ltd, 1989

        RUN file 'C' interface main program.
*/

char buf[2];                    /* this is BSS, unitialised. */

char *str="Hello world";        /* this is _DATA, initialised. */

unsigned int __brklvl;          /* required by Turboc library. */

int main()
    {
    puts(str);                  /* print initialised hello. */

    buf[0]='!';                 /* initialise uninitialised data. */
    buf[1]=0;

    puts(&buf[0]);              /* print uninitialised data. */

    return(0);
    }

void _exit()
/*
Dummy exit function required for the Turboc libraries.
*/
{}
```

```
TITLE   RU_C.ASM

comment *
        Copyright DIP Ltd., 1989

        'C' header for creation of .RUN files.

        Memory usage:

        ---------------------------------------- High memory -------
        SP:             Stack
        ----------------------------------------------------------
                        Uninitialised data
        ----------------------------------------------------------
        DS/SS:          Initialised data
        ----------------------------------------------------------
        ES:             PSP
        ---------------------------------------- Low memory --------
        *

;       Segment and Group declarations

; code and fixed data (less than 64k).
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
_TEXT   ENDS

; code ends (marker segment)
_TEXTEND   SEGMENT BYTE PUBLIC 'CEND'
_TEXTEND   ENDS

; initialised data transferred into RAM.
_DATA   SEGMENT PARA PUBLIC 'DATA'
_DATA   ENDS

; uninitialised data which is allocated space in RAM.
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS

; uninitialised data end (marker segment).
_BSSEND SEGMENT BYTE PUBLIC 'STACK'
_BSSEND ENDS

DGROUP GROUP _DATA,_BSS,_BSSEND         ; dgroup is all data segments.

        ASSUME  CS:_TEXT, DS:DGROUP

        extrn   _main:near              ; main 'C' routine.

STACKSIZE       equ     128             ; stack size in bytes.

; At the start, SS, DS and ES all point to the program segment prefix.
; CS is a ptr into the memory card.

_TEXT   SEGMENT

        org     0                       ; ip is zero on entry.

start   proc    near                    ; near is irrelevant, use fn 4c to
                                        ; terminate.

        mov     dx,ds                   ; ensure DS:0 is ptr to 1st data byte.
        add     dx,10h
        mov     ds,dx

        mov     bx,STACKSIZE
        add     bx,offset DGROUP:edata  ; bx has bytes of reqd RAM.

        push    bx                      ; this will be stack ptr.

        shr     bx,1
        shr     bx,1
        shr     bx,1
        shr     bx,1
        add     bx,11h                  ; 10h for PSP, 1 for rounding.
```

```
        mov     ah,4ah
        int     21h                     ; reduce RAM to required RAM.
        jc      abort                   ; can't reduce.

        pop     bx                      ; get calc'd p back.
        mov     ax,ds
        mov     ss,ax                   ; stack is in RAM.
        mov     sp,bx

        push    ds
        pop     es                      ; target is allocated RAM after PSP.

        push    cs
        pop     ds                      ; source is memory card.

        mov     si, offset _TEXT:etext  ; get ptr to last byte in code.
        add     si, 0Fh                 ; round up to 1st byte in data.
        and     si, 0FFF0h              ; data is para aligned on the card.
                                        ; ds:si is ptr to start of init data.
        xor     di,di                   ; put data at 0 offs into alloc'd RAM.
                                        ; es:di is ptr to alloc'd RAM target.
        mov     cx, offset DGROUP:bdata ; get bytes in initialised data.
        inc     cx                      ; round up: ensure last byte is copied.
        shr     cx,1

        cld
        rep     movsw                   ; copy init data from memory card.

        push    es
        pop     ds                      ; DS back to ptr to RAM.

        mov     di,offset DGROUP:bdata  ; ptr to where uninit data goes in RAM.
        mov     cx,offset DGROUP:edata  ; ptr to end of all data.
        sub     cx,di                   ; calc bytes in BSS.
        xor     al,al                   ; clear to zero.
        rep     stosb

        call    _main                   ; invoke program.

        mov     ah,4ch                  ; terminate with main's return code.
        int     21h

abort:
        mov     ax,4c01h                ; abort with error.
        int     21h
start   endp

_TEXT   ENDS


_TEXTEND   SEGMENT BYTE PUBLIC 'CEND'
etext   label   byte            ;Last byte of text segment
_TEXTEND   ENDS

_DATA   SEGMENT
        public  _errno
_errno  dw      0
_DATA   ENDS

_BSS    SEGMENT
bdata   label   byte
_BSS    ENDS

_BSSEND SEGMENT
edata   label   byte
_BSSEND ENDS

        end     start
```

APPENDIX B DIAGRAM OF PORTFOLIO CHARACTER SET

(all numbers in decimal)



APPENDIX C Example peripheral design

As an illustration of a typical Atari Portfolio peripheral we have designed a peripheral which flashes an LED in time with the system tick. In order that this peripheral operates transparently to the user we have used a ROM extension.

Peripheral Specification

i) have a single LED which will toggle on a timer tick.
ii) have a PID of 64h.
iii) have all software on an extension ROM.
iv) only operate with the machine powered on.
v) be a terminating peripheral.
vi) the peripheral will power up with the LED off.

Hardware Design (see schematic)

See section 2.6 & 2.7 for more information.

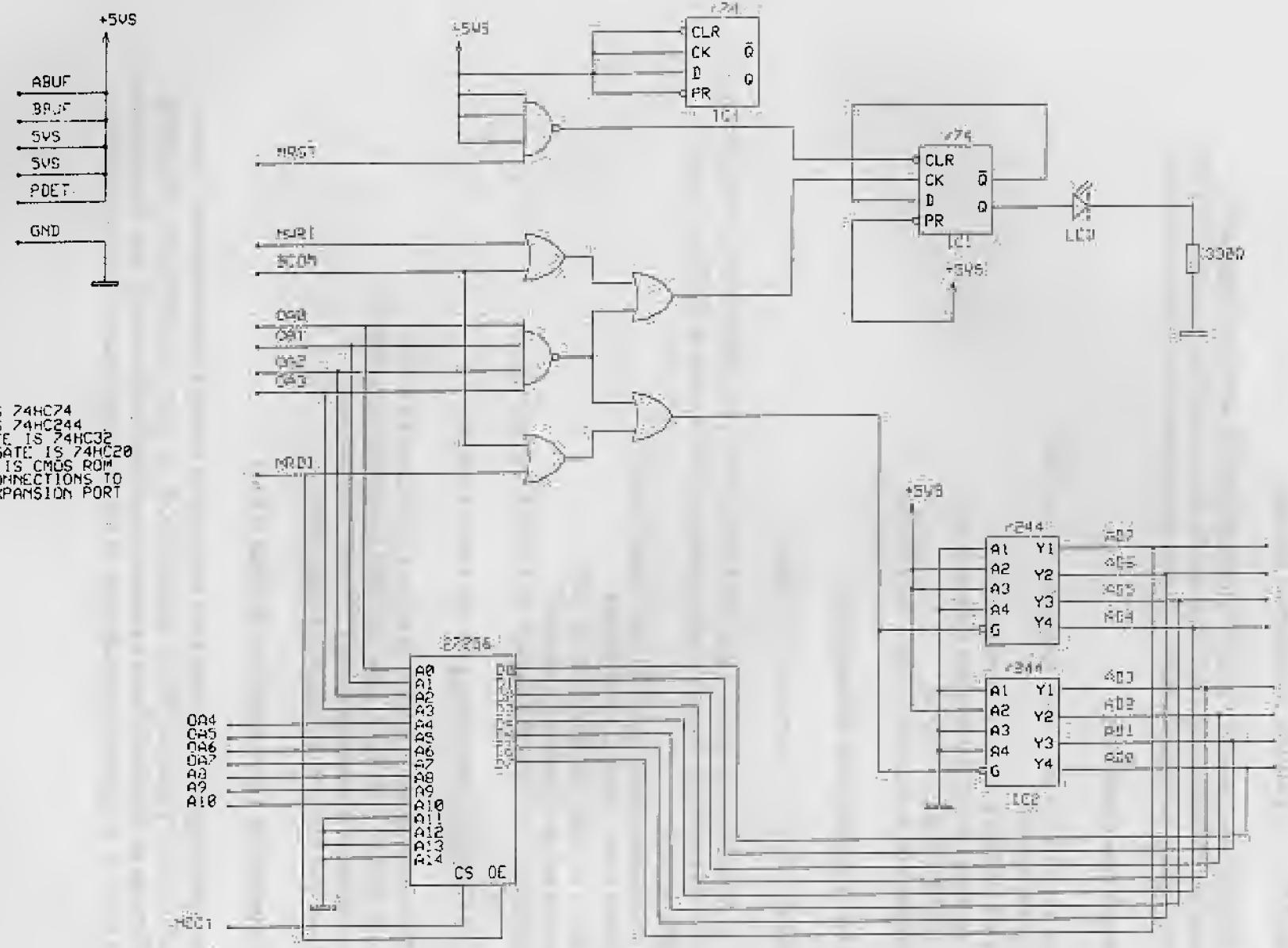
* There is decode logic to read the PID from 807Fh. (Since there is no need find out if the LED is on or off, the latch will be decoded for a write at 807Fh, to save decoding logic.)
* Each successive write to 807Fh will toggle the LED
* If the latch is set, the LED will be ON
* If the latch is reset, the LED will be OFF
* The circuit will be powered from 5VS
* The buffering signals ABUF/BBUF and PDET are also used

Software Design (see listing of file PDEM.ASM)

* A ROM extension will be used to hook a TSR onto INT 1Ch
* Each call to INT 1Ch will toggle the LED by writing to I/O 807Fh.
* If the peripheral is removed, the TSR will de-install itself.
* On power up, the LED is assumed to be reset.

ROM Extensions (see listing of XROM.ASM)

This example program illustrates how to design a simple ROM extension. It can either function as a specific BIOS, specific DOS, or Common extension, according to the ID code at 0C000:0. Each extension module identifies itself, and where it was invoked from. It is a good demonstration of the potential power of ROM extensions.



SCHEMATIC FOR EXAMPLE PERIPHERAL

```
;*****************************************************************
;       Module:         PDEM.ASM
;       Copyright:      DIP Ltd, 1989
;
;       Peripheral ROM Extension
;*****************************************************************

name            XROM

                assume  cs:cseg,ds:dseg

DOSX            equ     055aaH                  ;Specific DOS extension

cseg            segment

                org     0H                      ;Extension Vector table

xrom_main       label   near                    ;Start label for MASM

bixt_type       dw      DOSX                    ;Identification code

bixt_size       db      0                       ;Num 512 byte blocks in ROM

                org     3H
bixt_gdos       label   byte                    ;Specific DOS extension
                jmp     genx_vect               ;Specific extension vector

                org     40H
bixt_user       label   byte                    ;OEM user text
                db      'Crt Plant Periph'

                ;The plan is to allocate some memory, Copy a section
                ; of code to memory, and then point the Specified
                ; vector to that code.

xrom_proc       proc    FAR

INTR_NUMB       equ     1cH                     ;TIMER TICK

genx_vect       label   near

                push    ax                      ;Preserve registers
                push    bx                      ; required to set up
                push    ds                      ; local stack

                ;Allocate User RAM. Note that this can ONLY be done
                ; after DOS initialisation.

                mov     bx,(ALOC_SIZE+0fH)/10H  ;Paras to allocate
                mov     ah,48H                  ;Allocate memory
                int     21H

                mov     ds,ax                   ;Set DS to allocated RAM
                mov     stak_save,ss            ;Preserve Caller stack
                mov     stak_save+2,sp

                shl     bx,1                    ;Convert size to bytes
                shl     bx,1
                shl     bx,1
                shl     bx,1

                ;Set up User stack.

                mov     ss,ax                   ;Set up stack at top
                mov     sp,bx                   ; of allocated memory

                push    cx                      ;Preserve registers
                push    dx                      ; YOU MUST ALWAYS PRESERVE
                push    si                      ; ALL USER REGS
                push    di
                push    bp
                push    es
```

```
                ;Copy the ISR to the allocated area

                push    ds                      ;Preserve DS

                push    cs                      ;Set up Source
                pop     ds
                mov     si,offset tick_code

                push    ss                      ;Set up destination
                pop     es
                mov     di,offset load_base

                mov     cx,CODE_SIZE            ;Bytes to copy
                cld                             ;Initialise flag
                rep     movsb                   ;Copy TSR to RAM

                pop     ds                      ;Restore DS

                ;Get the specified vector, and set it to the ISR

                mov     ax,3500H+INTR_NUMB      ;Get current Int 1CH
                int     21H

                mov     tick_vect,bx            ;Preserve vector
                mov     tick_vect+2,es

                mov     ax,2500H+INTR_NUMB      ;Set interrupt vector
                mov     dx,offset load_base
                int     21H

                mov     ax,1501H                ;Generate Confidence BEEP
                int     61H

                pop     es                      ;Restore registers
                pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx

                mov     ss,stak_save            ;Restore Caller STACK
                mov     sp,stak_save+2

                pop     ds                      ;Restore remaining regs
                pop     bx
                pop     ax

                ret                             ;FAR return to caller
xrom_proc       endp

                ;Interrupt Service Routine (ISR)

PID_CODE        equ     64H                     ;Peripheral PID code
LEDS_PORT       equ     807fH                   ;LEDs I/O address

tick_code       label   byte                    ;TSR code

                push    ax                      ;Preserve registers
                push    dx
                push    bx
                push    ds
                push    es

                mov     ah,1aH                  ;Get Peripheral PID
                int     61H
                or      al,al                   ;Peripheral installed?
                jz      tick_none               ;No, so uninstall

                cmp     ah,PID_CODE             ;Correct peripheral?
                jne     tick_none               ;No, so uninstall

                ;Peripheral installed, so toggle LEDs

                mov     dx,LEDS_PORT            ;Toggle LED address
```

```
;*****************************************************************
;
;       Module:         PDLM.ASM
;       Copyright:      DIP Ltd. 1989
;
;       Peripheral ROM Extension
;
;*****************************************************************

name            XROM

                assume  cs:cseg,ds:dseg

DOSX            equ     055aaH                  ;Specific DOS extension

cseg            segment

                org     0H                      ;Extension Vector table

xrom_main       label   near                    ;Start label for MASM

bixt_type       dw      DOSX                    ;Identification code

bixt_size       db      0                       ;Num 512 byte blocks in ROM

                org     3H
bixt_gdos       label   byte                    ;Specific DOS extension
                jmp     genx_vect               ;Specific extension vector

                org     40H
bixt_user       label   byte                    ;OEM user text
                db      'Crt Plant Periph'

                ;The plan is to allocate some memory, Copy a section
                ; of code to memory, and then point the Specified
                ; vector to that code.

xrom_proc       proc    FAR

INTR_NUMB       equ     1cH                     ;TIMER TICK

genx_vect       label   near

                push    ax                      ;Preserve registers
                push    bx                      ; required to set up
                push    ds                      ; local stack

                ;Allocate User RAM. Note that this can ONLY be done
                ; after DOS initialisation.

                mov     bx,(ALOC_SIZE+0fH)/10H  ;Paras to allocate
                mov     ah,48H                  ;Allocate memory
                int     21H

                mov     ds,ax                   ;Set DS to allocated RAM
                mov     stak_save,ss            ;Preserve Caller stack
                mov     stak_save+2,sp

                shl     bx,1                    ;Convert size to bytes
                shl     bx,1
                shl     bx,1
                shl     bx,1

                ;Set up User stack.

                mov     ss,ax                   ;Set up stack at top
                mov     sp,bx                   ; of allocated memory

                push    cx                      ;Preserve registers
                push    dx                      ; YOU MUST ALWAYS PRESERVE
                push    si                      ; ALL USER REGS
                push    di
                push    bp
                push    es
```

```
                ;Copy the ISR to the allocated area

                push    ds                      ;Preserve DS

                push    cs                      ;Set up Source
                pop     ds
                mov     si,offset tick_code

                push    ss                      ;Set up destination
                pop     es
                mov     di,offset load_base

                mov     cx,CODE_SIZE            ;Bytes to copy
                cld                             ;Initialise flag
                rep     movsb                   ;Copy TSR to RAM

                pop     ds                      ;Restore DS

                ;Get the specified vector, and set it to the ISR

                mov     ax,3500H+INTR_NUMB      ;Get current Int 1CH
                int     21H

                mov     tick_vect,bx            ;Preserve vector
                mov     tick_vect+2,es

                mov     ax,2500H+INTR_NUMB      ;Set interrupt vector
                mov     dx,offset load_base
                int     21H

                mov     ax,1501H                ;Generate Confidence BEEP
                int     61H

                pop     es                      ;Restore registers
                pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx

                mov     ss,stak_save            ;Restore Caller STACK
                mov     sp,stak_save+2

                pop     ds                      ;Restore remaining regs
                pop     bx
                pop     ax

                ret                             ;FAR return to caller
xrom_proc       endp

                ;Interrupt Service Routine (ISR)

PID_CODE        equ     64H                     ;Peripheral PID code
LEDS_PORT       equ     807fH                   ;LEDs I/O address

tick_code       label   byte                    ;TSR code

                push    ax                      ;Preserve registers
                push    dx
                push    bx
                push    ds
                push    es

                mov     ah,1aH                  ;Get Peripheral PID
                int     61H
                or      al,al                   ;Peripheral installed?
                jz      tick_none               ;No, so uninstall

                cmp     ah,PID_CODE             ;Correct peripheral?
                jne     tick_none               ;No, so uninstall

                ;Peripheral installed, so toggle LEDs

                mov     dx,LEDS_PORT            ;Toggle LED address
```

```
                out     dx,al                   ;AL unimportant
                jmp     short tick_exit         ;Exit
tick_none:
                assume  cs:dseg                 ;Force DSEG offset

                ;Invalid Peripheral, so uninstall TSR

                mov     ax,2500H+INTR_NUMB      ;Set interrupt vector
                mov     bx,offset tick_vect     ;Get old vector
                mov     ds,cs:[bx+2]
                mov     dx,cs:[bx]
                int     21H

                ;Now vector reset, free allocated memory

                push    cs                      ;Segment of block
                pop     es
                mov     ah,49H                  ;Free memory
                int     21H
tick_exit:
                pop     es                      ;Restore registers
                pop     ds
                pop     dx
                pop     bx
                pop     ax

                jmp     dword ptr cs:tick_vect  ;Jump to old TSR

CODE_SIZE       equ     $-tick_code             ;Size of ISR

cseg            ends

                ;Data segment TEMPLATE (No initialised data here!)

dseg            segment

data_sptr       label   byte                    ;Start of Data

stak_save       dw      ?                       ;Caller stack stored here
                dw      ?

tick_vect       dw      ?                       ;Old vector stored here
                dw      ?

load_base       label   byte                    ;Start of ISR

LOAD_SIZE       equ     ($-data_sptr)+CODE_SIZE ;Load module size
ALOC_SIZE       equ     LOAD_SIZE+100H          ;Load module + Stack

dseg            ends

                end     xrom_main
```

```
;*****************************************************************
;       Module:         XROM.ASM
;       Copyright:      DIP Ltd. 1989
;
;       ROM Extension DEMO program
;
;
;       A ROM extension may be run from a Credit Card Memory
;       or an Extension ROM.
;
;       The Extension code must preserve ALL registers!
;
;       The Pre-BIOS vector MUST return by a FAR JMP to
;       OFFFE:0, as no stack is set up at this stage
;*****************************************************************

name            XROM

                assume  cs:cseg

LF              equ     0aH                     ;Line feed
CR              equ     0dH                     ;Carriage return

BIOX            equ     0aa55H                  ;Specific BIOS extension
DOSX            equ     055aaH                  ;Specific DOS extension
BIDO            equ     05555H                  ;Complete control

cseg            segment

                org     0H                      ;Extension Vector table

xrom_main       label   near

bixt_type       dw      BIOX                    ;Identification code

bixt_size       db      0                       ;Num 512 byte blocks in ROM

                org     3H
bixt_gbio       label   byte                    ;Specific BIOS extension
bixt_gdos       label   byte                    ;Specific DOS extension
                jmp     genx_vect               ;Specific extension vector

                org     40H
bixt_user       label   byte                    ;OEM user text
                db      'Test ROM (C) DIP'

                org     50H
bixt_preb:      jmp     preb_vect               ;Pre-bios jmp vector

                org     55H
bixt_bext:      jmp     bext_vect               ;Bios-ext jmp vector

                org     5aH
bixt_pdos:      jmp     pdos_vect               ;Pre-dos jmp vector

                org     5fH
bixt_dext:      jmp     dext_vect               ;Dos-ext jmp vector

                org     64H
bixt_ados:      jmp     ados_vect               ;Post-dos jmp vector

                org     69H
bixt_pwdn:      jmp     pwdn_vect               ;Power down jmp vector

                org     6eH
bixt_pwup:      jmp     pwup_vect               ;Power up jmp vector

xrom_proc       proc    FAR

                ;Determine extension type

genx_vect       label   near
```

```
                push    bp                      ;Preserve BP

                cmp     cs:[0],BIOX             ;Specific BIOS extension ?
                jne     not_genb                ;No

                mov     bp,offset gbio_text     ;Specific BIOS extn text
                jmp     short xrom_disp
not_genb:
                cmp     cs:[0],DOSX             ;Specific DOS extension ?
                jne     not_gend                ;No

                mov     bp,offset gdos_text     ;Specific DOS extn text
                jmp     short xrom_disp
not_gend:
                mov     bp,offset invl_text     ;Invalid text
                jmp     short xrom_disp


preb_vect       label   near

                jmp     dword ptr cs:preb_retn  ;Pre-BIOS extension

preb_retn       dw      0
                dw      0fffeH

bext_vect       label   near                    ;Post-BIOS extension

                push    bp
                mov     bp,offset bext_text
                jmp     short xrom_disp

pdos_vect       label   near                    ;Pre-DOS extension

                push    bp
                mov     bp,offset pdos_text
                jmp     short xrom_disp

dext_vect       label   near                    ;DOS extension

                push    bp
                mov     bp,offset dext_text
                jmp     short xrom_disp

ados_vect       label   near                    ;Post-DOS extension

                push    bp
                mov     bp,offset ados_text
                jmp     short xrom_disp

pwdn_vect       label   near                    ;Power-Down extension

                push    bp
                mov     bp,offset pwdn_text
                jmp     short xrom_disp

pwup_vect       label   near                    ;Power-Up extension

                push    bp
                mov     bp,offset pwup_text
                jmp     short xrom_disp

xrom_disp       label   near                    ;Main display routine

                push    ax                      ;Preserve registers
                push    bx
                push    cx
                push    dx
                push    es

                call    disp_text               ;Display text in BP

                mov     ax,2400H                ;Get ROM state
                int     61H

                or      dl,dl                   ;Normal ROM ?
```

```
                jnz     not_norm                ;No, so skip

                mov     bp,offset norm_text     ;Get normal ROM text
                jmp     short stat_disp
not_norm:
                dec     dl                      ;Drive A ?
                jnz     not_drva                ;No, so skip

                mov     bp,offset drva_text     ;Get Drive A text
                jmp     short stat_disp
not_drva:
                dec     dl                      ;Drive B ?
                jnz     not_drvb                ;No, so skip

                mov     bp,offset drvb_text     ;Get Drive B text
                jmp     short stat_disp
not_drvb:
                dec     dl                      ;Drive B ?
                jnz     not_xrom                ;No, so skip

                mov     bp,offset xrom_text     ;Get Drive B text
                jmp     short stat_disp
not_xrom:
                mov     bp,offset invl_text
stat_disp:
                call    disp_text               ;Display text in BP

                mov     bp,offset crlf_text     ;Finally CR, LF
                call    disp_text

                pop     es                      ;Restore registers
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                pop     bp

                ret                             ;FAR return
xrom_proc       endp

;*****************************************************************;
;                                                                 ;
;       Main Display routine                                      ;
;                                                                 ;
;*****************************************************************;

disp_text       proc    near

                xor     bh,bh                   ;Page 0
                mov     ah,3                    ;Get cursor position in DX
                int     10H

                push    cs                      ;Access text
                pop     es

                xor     ch,ch                   ;Initialise
                mov     cl,es:[bp]              ;Get length
                inc     bp                      ;Advance to text

                mov     ax,1301H                ;Write string
                int     10H

                ret

disp_text       endp

gbio_text       db      gdos_text-$-1
                db      'Spec BIOS Extension - '

gdos_text       db      bext_text-$-1
                db      'Spec DOS  Extension - '

bext_text       db      pdos_text-$-1
                db      'Com BIOS Extension - '
```

```
pdos_text       db      dext_text-$-1
                db      'Pre-DOS  Extension -
dext_text       db      ados_text-$-1
                db      'Com DOS  Extension -
ados_text       db      pwdn_text-$-1
                db      'Post-DOS Extension -
pwdn_text       db      pwup_text-$-1
                db      'Power Down Extension -
pwup_text       db      norm_text-$-1
                db      'Power Up Extension -
norm_text       db      drva_text-$-1
                db      'Normal ROM'
drva_text       db      drvb_text-$-1
                db      'CCM Drive A'
drvb_text       db      xrom_text-$-1
                db      'CCM Drive B'
xrom_text       db      invl_text-$-1
                db      'Extn ROM'
invl_text       db      crlf_text-$-1
                db      'Invalid'
crlf_text       db      2,CR,LF
cseg            ends
                end     xrom_main
```

APPENDIX D Memory size and assignment

REDUCING STACK OR MEMORY SIZE

Some compiled (.EXE) programs require reduced stack or memory size requirements when designed for the Portfolio. Reduced stack or memory size requirements are also needed when executing some built in applications.

Microsoft's Assembler includes a utility (EXEMOD) which allows modification of maximum required memory word in the .EXE header. Modifying the header allows the applications room in which to execute.

MEMORY ASSIGNMENT

Portfolio software developers should include the following interrupt information in a header:

INT 21 fn 4ah

This DOS function reduces memory used to the amount required by the program, rather than defaulting to all memory.